

NAVAL POSTGRADUATE SCHOOL Monterey, California



PHOENIX AUTONOMOUS UNDERWATER VEHICLE RECOVERY SOFTWARE REFERENCE

by

Duane Davis
Don Brutzman
Robert McGhee

September 1996

19961028 025

Approved for public release; distribution is unlimited.

Prepared for: Naval Postgraduate School
Monterey, CA 93943

DTIC QUALITY INSPECTED 1

DISCLAIMER NOTICE



**THIS DOCUMENT IS BEST
QUALITY AVAILABLE. THE
COPY FURNISHED TO DTIC
CONTAINED A SIGNIFICANT
NUMBER OF PAGES WHICH DO
NOT REPRODUCE LEGIBLY.**

NAVAL POSTGRADUATE SCHOOL
Monterey, California

REAR ADMIRAL M.J. EVANS
Superintendent

RICHARD S. ELSTER
Provost

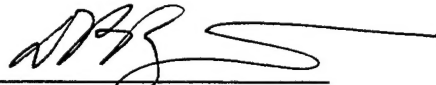
This report was prepared for the Naval Postgraduate School

Reproduction of all or part of this report is authorized.

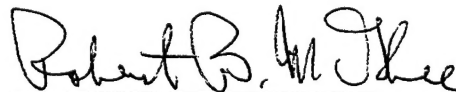
This report was prepared by:



Duane Davis
Lieutenant, United States Navy



Don Brutzman
Assistant Professor
Undersea Warfare Group



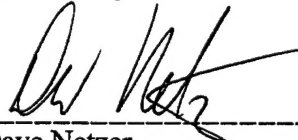
Robert B. McGhee
Professor of Computer Science

Reviewed by:



Ted Lewis
Chairman

Released by:



Dave Netzer
Dean of Research

REPORT DOCUMENTATION PAGE

Form approved

OMB No 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave blank)**2. REPORT DATE**

September 1996

3. REPORT TYPE AND DATES COVERED

Technical Report: September 1996

4. TITLE AND SUBTITLE

Phoenix Autonomous Underwater Vehicle Recovery Software Reference

5. FUNDING

N/A

6. AUTHOR(S)

Duane Davis, Don Brutzman, and Robert McGhee

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Postgraduate School
Department of Computer Science
Monterey, CA 93943-5000

**8. PERFORMING ORGANIZATION
REPORT NUMBER**

NPS-CS-96-008

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Naval Postgraduate School
Monterey, CA 93943-5000

**10. SPONSORING/MONITORING
AGENCY REPORT NUMBER****11. SUPPLEMENTARY NOTES**

The views expressed in this report are those of the author and do not reflect the official policy or position of the Department of Defense or the United States Government.

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

A

13. ABSTRACT (Maximum 200 words.)

Because of range limitations imposed by speed and power supplies, convert launch and recovery of Autonomous Underwater Vehicles (AUVs) near the operating area will be required for their use in many military applications. This report contains source code implemented on the Phoenix AUV in support of recovery in a small stationary tube.

Implementation involves the development of low-level behaviors for sonar and vehicle control, mil-level tactics for recovery planning, and a mission-planning system for translating high-level goals into an executable mission. Sonar behaviors consist of modes for locating and tracking objects, while vehicle control behaviors include the ability to drive to and maintain a position relative to a tracked object. Finally, a mission-planning system allowing graphical specification of mission objectives and recovery parameters is implemented.

14. SUBJECT TERMS

Sonar behaviors, autonomous underwater vehicles

**15. NUMBER OF
PAGES**

310 pages

16. PRICE CODE**17. SECURITY CLASSIFICATION
OF REPORT**

Unclassified

**18. SECURITY CLASSIFICATION
OF THIS PAGE**

Unclassified

**19. SECURITY CLASSIFICATION
OF ABSTRACT**

Unclassified

**20. LIMITATION OF
ABSTRACT**
SAR

ABSTRACT

Because of range limitations imposed by speed and power supplies, covert launch and recovery of Autonomous Underwater Vehicles (AUVs) near the operating area will be required for their use in many military applications. This report contains source code implemented on the *Phoenix* AUV in support of recovery in a small stationary tube.

Implementation involves the development of low-level behaviors for sonar and vehicle control, mid-level tactics for recovery planning, and a mission-planning system for translating high-level goals into an executable mission. Sonar behaviors consist of modes for locating and tracking objects, while vehicle control behaviors include the ability to drive to and maintain a position relative to a tracked object. Finally, a mission-planning system allowing graphical specification of mission objectives and recovery parameters is implemented.

TABLE OF CONTENTS

I.	INTRODUCTION	1
II.	UNDERWATER VIRTUAL WORLD SOURCE CODE	3
III.	EXECUTION LEVEL SOURCE CODE	61
IV.	TACTICAL LEVEL SOURCE CODE	241
V.	MISSION GENERATION EXPERT SYSTEM SOURCE CODE	259
	LIST OF REFERENCES	303
	INITIAL DISTRIBUTION LIST	305

I. INTRODUCTION

This technical report contains *Phoenix* Autonomous Underwater Vehicle (AUV) source code that was developed during the conduct of research in support of recovery in a small tube [Davis 96]. The technical report is divided into four sections. The first section consists Underwater Virtual World (UWV) [Brutzman 94] source code that was developed to support sonar simulation and visualization. The second section consists of execution level software that was developed to provide low level behaviors such as sonar target tracking and station keeping. The third section consists of tactical level software that implements higher level tactics for recovery path planning and execution level command generation. The final section consists of source code for a mission planning expert system that was developed to support rapid executable mission generation. The source code included in this technical report does not include all *Phoenix* source code, but rather only that code that was modified or developed during the conduct of this research. This code is compiled and linked with existing *Phoenix* software to implement the Rational Behavior Model software architecture [Byrnes 92]. All source code contained in this technical report is available online individually or as part of the *Phoenix* AUV software package in .tar format at

http://www.stl.nps.navy.mil/~brutzman/dissertation/software_reference.html

II. UNDERWATER VIRTUAL WORLD SOURCE CODE

This section contains source code for the UVW. Included code is only that code which was written or modified during the conduct of this thesis. The file viewer.C must be compiled and linked with the file AUVglobals.H to make the viewer portion of the UVW. The file ivSonar.C must be compiled and linked with the files dynamics.C, UUVBody.C, AUVsocket.C, DISNetworkedRigidBody.C, RigidBody.C, Hmatrix.C, Quaternion.C, Vector3d.C, AUVglobals.H, UUVmodel.H and AUVmodel.H to make the dynamics module of the UVW. These files and installation instructions and examples are available online at

http://www.stl.nps.navy.mil/~brutzman/dissertation/software_reference.html

Files are available individually or as a group in a .tar package.

```

////////////////////////////////////
/*
Program:      viewer.C

Description:   OpenInventor viewer for
               NPS AUV Underwater Virtual World

Author:       Don Brutzman, Duane Davis

Revised:      25 June 96

System:       Irix 5.3

Compiler:     ANSI C++ / OpenInventor API

Compilation:  irix> make viewer

References: (1) IEEE Protocols for Distributed Interactive Simulation (DIS)
               Applications version 2.0, Institute for Simulation and
               Training, Universit of Central Florida, Orlando Florida,
               28 May 1993.

               (2) Macedonia, Michael, Zeswitz, Steven, and Locke, John,
               Distributed Interactive Simulation (DIS) multicast
               version 2.0.3, Naval Postgraduate School, February 94.

               (3) Zeswitz, Steven, "NPSNET: Integration of Distributed
               Interactive Simulation (DIS) Protocol for Communication
               Architecture and Information Interchange." master's thesis,
               Naval Postgraduate School, Monterey California, 28 May 1993.

               (4) Wernecke, Josie and the OpenInventor Architecture Group,
               _The Inventor Mentor_, Addison-Wesley, Reading Massachusetts,
               1994.

Dissertation: Brutzman, Donald P., A Virtual World for an Autonomous
               Underwater Vehicle, Ph.D. Dissertation, Naval Postgraduate
               School, Monterey California, December 1994. Available at
               http://www.stl.nps.navy.mil/~brutzman/dissertation/

               Brutzman, Donald P., Software Reference: A Virtual World
               for an Autonomous Underwater Vehicle, technical report
               NPS-CS-010-94, Naval Postgraduate School, Monterey
               California, December 1994. The accompanying public
               electronic distribution of this reference includes source
               code and executable programs. World-Wide Web (WWW)
               Uniform Resource Locator (URL) is
               ftp://taurus.cs.nps.navy.mil/pub/auv/auv_uvw.html

Advisors:     Dr. Mike Zyda, Dr. Bob McGhee and Dr. Tony Healey

Notes:        Underwater virtual world is in feet, standard DIS PDU is
               meters

               Inventor 1.0.1 and DIS libraries were NOT compatible due to
               programs hangups which are triggered by mallocs in the
               DIS libraries conflicting with the Inventor window.
               Removing DIS libraries and just putting malloc's in the
               Inventor screen redraw callback function caused identical
               program hangups. No fix was found. Upgrading to OpenInventor
               under Irix 5.2 eliminated this problem completely.

               Automated camera control fixed.

Future work:  .iv file load via WWW URLs passed by DIS Message PDUs

```

Message PDUs including sound/speech.
 Add CAMERA_FROM_SONAR and camera selection menu.
 Get full scene optimization by ivquick fixed.
 Ocean current and DiveTracker range visualization.
 Optimization.
 Keyboard or button interactivity (using SceneViewer?)
 ivfix gives bad output ?!

```

*/
/////////////////////////////////////////////////////////////////
// from relnotes inventor_dev 4 (version 2.1)
#include <inttypes.h>

#include "../dynamics/AUVglobals.H"

#include <iostream.h>
#include <fstream.h>
#include <iomanip.h> // must follow iostream.h
#include <string.h>
#include <math.h>
#include <time.h>
#include <getopt.h>
#include <stdlib.h>
#include <netdb.h>
#include <netinet/in.h>
#include <sys/types.h>

/////////////////////////////////////////////////////////////////
//
// DIS includes. See Makefile for other DIS #include files; they must match.

#include "../DIS.mcast/h/disdefs.h"
#include "../DIS.mcast/h/appearance.h"

extern "C" { printPDU (char *); }; // function prototype provided for
                                   // compatibility, missing from DIS library

/////////////////////////////////////////////////////////////////
//
#include <X11/Intrinsic.h>
#include <X11/keysym.h>
#include <Xm/Xm.h>
#include <Xm/CascadeBG.h>
#include <Xm/Form.h>
#include <Xm/RowColumn.h>
#include <Xm/PushButton.h>
#include <Xm/PushButtonBG.h>
#include <Xm/Separator.h>
#include <Xm/SeparatorG.h>
#include <Xm/ToggleB.h>
#include <Xm/ToggleBG.h>

#include <Inventor/SbBasic.h>
#include <Inventor/SbViewportRegion.h>
#include <Inventor/So.h>
#include <Inventor/SoDB.h>
#include <Inventor/SoInput.h>
  
```

```

#include <Inventor/SoPickedPoint.h>
#include <Inventor/Xt/SoXt.h>
#include <Inventor/Xt/SoXtPrintDialog.h> // see SoOffscreenRender
#include <Inventor/Xt/SoXtRenderArea.h>
#include <Inventor/Xt/SoXtMaterialEditor.h>
#include <Inventor/Xt/viewers/SoXtExaminerViewer.h>
#include <Inventor/Xt/viewers/SoXtFlyViewer.h>
#include <Inventor/Xt/viewers/SoXtPlaneViewer.h>
#include <Inventor/Xt/viewers/SoXtWalkViewer.h>

#include <Inventor/actions/SoAction.h>
#include <Inventor/actions/SoCallbackAction.h>
#include <Inventor/actions/SoGLRenderAction.h>
#include <Inventor/actions/SoPickAction.h>
#include <Inventor/actions/SoRayPickAction.h>
#include <Inventor/actions/SoWriteAction.h>

#include <Inventor/engines/SoCalculator.h>
#include <Inventor/engines/SoElapsedTime.h>
#include <Inventor/engines/SoTimeCounter.h>

#include <Inventor/events/SoEvent.h>
#include <Inventor/events/SoKeyboardEvent.h>

#include <Inventor/nodes/SoComplexity.h>
#include <Inventor/nodes/SoCone.h>
#include <Inventor/nodes/SoCoordinate3.h>
#include <Inventor/nodes/SoCube.h>
#include <Inventor/nodes/SoCylinder.h>
#include <Inventor/nodes/SoCone.h>
#include <Inventor/nodes/SoDirectionalLight.h>
#include <Inventor/nodes/SoDrawStyle.h>
#include <Inventor/nodes/SoEventCallback.h>
#include <Inventor/nodes/SoFaceSet.h>
#include <Inventor/nodes/SoGroup.h>
#include <Inventor/nodes/SoMaterial.h>
#include <Inventor/nodes/SoNurbsSurface.h>
#include <Inventor/nodes/SoPerspectiveCamera.h>
#include <Inventor/nodes/SoPickStyle.h>
#include <Inventor/nodes/SoRotationXYZ.h>
#include <Inventor/nodes/SoScale.h>
#include <Inventor/nodes/SoSelection.h>
#include <Inventor/nodes/SoSeparator.h>
#include <Inventor/nodes/SoSphere.h>
#include <Inventor/nodes/SoSwitch.h>
#include <Inventor/nodes/SoTransform.h>
#include <Inventor/nodes/SoTransformSeparator.h>
#include <Inventor/nodes/SoTranslation.h>
#include <Inventor/nodes/SoTriangleStripSet.h> // order matters here
#include <Inventor/nodes/SoUnits.h>

#include <Inventor/sensors/SoTimerSensor.h>

////////////////////////////////////
///
// function prototypes

static int  DIS_net_open  ();

static void DIS_net_close ();

static void DIS_Redraw_Callback ( void      * unused_data,
                                   SoSensor * unused_calling_sensor );

double auv_ST725_range ( double angle, double power, double range,
                        SbViewportRegion * viewPort );

```



```

void readCircles ();

/////////////////////////////////////////////////////////////////

#define PI      3.1415926535897932

#define METERS_PER_FT  0.3048
#define FT_PER_METERS  3.2808

int      DEADRECKON      = TRUE;
int      READCIRCLEFILE  = FALSE;

ifstream circleFile;

// utility function prototypes

void      swap_float (float a, float b);

double    sign           (double x);

double    degrees        (double x); // radians input
double    radians        (double x); // degrees input
double    arcclamp       (double x);

double    dnormalize      (double angle_radians); // returns 0..2PI
int       inormalize      (double angle_radians) // returns 0..359
                        {return (int) (degrees (angle_radians) + 0.5) % 360;}

SoSeparator * readFile(const char *filename); // Inventor Mentor p. 284

void      initialize_globals ();

void      parse_command_line_flags (int argc, char ** argv);

/////////////////////////////////////////////////////////////////
// global data (referenced in callback routines, thus defined here)

#ifndef VIEWER_GLOBALS
#define VIEWER_GLOBALS // prevent warnings if multiple #includes present

SoSeparator * root;
SoSeparator * scenery = new SoSeparator;
// permit multiple loads from command line
SoTransform * AUV_position_node;

// initializers needed to avoid startup segmentation fault

SoRotationXYZ * rotate_AUV_z = new SoRotationXYZ;
SoRotationXYZ * rotate_AUV_y = new SoRotationXYZ;
SoRotationXYZ * rotate_AUV_x = new SoRotationXYZ;

SoRotationXYZ * rotate_AUV_bow_rudders = new SoRotationXYZ;
SoRotationXYZ * rotate_AUV_stern_rudders = new SoRotationXYZ;
SoRotationXYZ * rotate_AUV_bow_planes = new SoRotationXYZ;
SoRotationXYZ * rotate_AUV_stern_planes = new SoRotationXYZ;

SoCone * thrusterWakeFV = new SoCone;
SoCone * thrusterWakeAV = new SoCone;
SoCone * thrusterWakeFH = new SoCone;
SoCone * thrusterWakeAH = new SoCone;
SoCone * conePropellerWakeStbd = new SoCone;
SoCone * conePropellerWakePort = new SoCone;

SoSwitch * whichWakeFV = new SoSwitch;
SoSwitch * whichWakeAV = new SoSwitch;

```

```

SoSwitch      * whichWakeFH      = new SoSwitch;
SoSwitch      * whichWakeAH      = new SoSwitch;

SoTransform   * topsideBow        = new SoTransform;
SoTransform   * topsideStern      = new SoTransform;
SoTransform   * bottomsideBow     = new SoTransform;
SoTransform   * bottomsideStern   = new SoTransform;
SoTransform   * leftsideBow       = new SoTransform;
SoTransform   * leftsideStern     = new SoTransform;
SoTransform   * rightsideBow      = new SoTransform;
SoTransform   * rightsideStern    = new SoTransform;

SoTransform   * xfConeSonarST1000 = new SoTransform;
SoTransform   * offsetST1000      = new SoTransform;
SoRotationXYZ * rotConeSonarST1000 = new SoRotationXYZ;
SoCone        * coneSonarST1000   = new SoCone;

SoTransform   * xfConeSonarST725  = new SoTransform;
SoTransform   * offsetST725       = new SoTransform;
SoRotationXYZ * rotConeSonarST725  = new SoRotationXYZ;
SoScale       * scaleConeSonarST725 = new SoScale;
SoCone        * coneSonarST725     = new SoCone;

SoMaterial * silver = new SoMaterial;
SoMaterial * gold   = new SoMaterial;
SoMaterial * brass  = new SoMaterial;
SoMaterial * chrome = new SoMaterial;
SoMaterial * npsblue = new SoMaterial;
SoMaterial * seagreen = new SoMaterial;
SoMaterial * seasurface = new SoMaterial;
SoMaterial * seacolor = new SoMaterial;
SoMaterial * darkgreen = new SoMaterial;
SoMaterial * sonar_cone_color725 = new SoMaterial;
SoMaterial * sonar_cone_color1000 = new SoMaterial;

SoDrawStyle * wires = new SoDrawStyle;

SoScale * MetersToFeet = new SoScale;
SoScale * InchesToFeet = new SoScale;
SoScale * FeetToInches = new SoScale;

SoTransform * MineFieldTransform = new SoTransform;
SoScale     * MineFieldScale     = new SoScale;

SbViewportRegion * myRegion;

#define CAMERA_FREE      0
#define CAMERA_TO_AUV    1
#define CAMERA_FROM_AUV  2
#define CAMERA_FROM_SONAR 3

static int      whichCamera      = CAMERA_TO_AUV; // default camera

static int      PRINTDIALOG      = FALSE;

static int      BACKGROUNDCOLORDIALOG = FALSE;

static int      TEXTURE          = FALSE;

static int      SCREENDOOR       = FALSE;

static int      LIGHTSOUT        = FALSE;

static int      MINEFIELD        = FALSE;

SoPerspectiveCamera * PerspectiveCameraFree = new SoPerspectiveCamera;
SoPerspectiveCamera * PerspectiveCameraToAUV = new SoPerspectiveCamera;
SoPerspectiveCamera * PerspectiveCameraFromAUV = new SoPerspectiveCamera;

```

```

SoKeyboardEvent      * key_event                      = new SoKeyboardEvent;

SbVec3f      behindAUVPosition;
SbVec3f      aheadOfAUVPosition;
SbVec3f      priorAUVPosition;
SbVec3f      currentAUVPosition;

SbVec3f      behindSonarPosition;
SbVec3f      aheadOfSonarPosition;

SbVec3f      priorCameraPosition;
SbVec3f      currentCameraPosition;
SbVec3f      priorCameraOffset;
SbVec3f      currentCameraOffset;

SbVec3f      orientationRotationAxis;
float         orientationRotationAngle;

SbVec3f      standardCameraOffset      = SbVec3f ( 3.0, 1.0, 6.0 );
SbVec3f      AUVCameraOffset           = SbVec3f ( 0.0, 1.2, 0.0 );
SbVec3f      AUVCameraPointTo         = SbVec3f ( +3.0, 0.0, 0.0 );
float         standardCameraFocalDistance = 20.0;

SoSeparator * command_line_node;

static clock_t      time_stamp_of_current_PDU,
                    time_of_PDU_receipt,
                    current_clock,
                    delta_clock;
static double       delta_time;

static int          PDU_overdue;

static int          DIS_port_open;

char               port [ 6];
char               group [30];

static SoUnits      * unitsfeet;

// static struct in_addr      inaddr;

// static struct hostent      * hp;

double ST1000Complexity_off = 0.05;
double ST1000Complexity_on  = 0.2;
double ST725Complexity_off  = 0.1;
double ST725Complexity_on   = 0.4;

SoComplexity * wakeComplexity = new SoComplexity;
SoComplexity * ST1000Complexity = new SoComplexity;
SoComplexity * ST725Complexity = new SoComplexity;

////////////////////////////////////
// all NPS AUV dimensions here in inches

#define HULLBODYLENGTH 54.00
#define HULLBODYWIDTH 16.50
#define HULLBODYHEIGHT 10.0

#define SEAM -27.00 // - HULLBODYLENGTH / 2.0
#define STERN -43.50 // - HULLBODYLENGTH / 2.0 - 16.5
#define LEFT 8.25 // HULLBODYWIDTH / 2.0

```

```

#define RIGHT            -8.25    // - HULLBODYWIDTH / 2.0
#define TOP              5.00    // HULLBODYHEIGHT / 2.0
#define BOTTOM           -5.00    // - HULLBODYHEIGHT / 2.0

#define DOMECONTROLPT    25.00    // nosecone NURBS surface coordinates
#define DOMECONTROLPTHALF 20.00    // offsets to help define shape

#define TOPHALF          4.75
#define BOTTOMHALF       -4.75
#define LEFT_HALF        8.00
#define RIGHTHALF        -8.00
#define CENTER           0.00

#define FINLENGTH        6.00    // fins need to be tapered
#define FINWIDTH         0.75    // fin thickness 1" at bottom, 0.5" at top
#define FINHEIGHT        6.75

#define FINOFFSETFORWARD 24.0
#define FINOFFSETAFT     -33.0
#define FINOFFSETLEFT    12.10 // (HULLBODYWIDTH / 2) + (FINHEIGHT / 2) + .5"
#define FINOFFSETRIGHT   -12.10
#define FINOFFSETUP      8.875 // (HULLBODYHEIGHT / 2) + (FINHEIGHT / 2) + .5"
#define FINOFFSETDOWN    -8.875

#define THRUSTERID       3.0
#define THRUSTEROD       3.5
#define THRUSTERFORWARDV 13.0    // forward SEAM - 14
#define THRUSTERAFTV     -21.0   // SEAM + 6
#define THRUSTERFORWARDH 19.0    // forward SEAM - 8
#define THRUSTERAFTH     -26.0   // SEAM + 1

#define SHAFTOFFSETLEFT  3.75
#define SHAFTOFFSETRIGHT -3.75

// reverify these dimensions after rebuild
#define CARDCAGELEFT     4.00
#define CARDCAGERIGHT    -4.00
#define CARDCAGELENGTH   12.00
#define CARDCAGEWIDTH    7.00
#define CARDCAGEHEIGHT   8.00

////////////////////////////////////

#endif    //    VIEWER_GLOBALS

////////////////////////////////////
SoSeparator * makeAUV ()
{
    rotate_AUV_z->angle.setValue ( AUV_psi);
    rotate_AUV_z->axis.setValue (SoRotationXYZ::Z);

    rotate_AUV_y->angle.setValue (- AUV_theta);
    rotate_AUV_y->axis.setValue (SoRotationXYZ::Y);

    rotate_AUV_x->angle.setValue ( AUV_phi);
    rotate_AUV_x->axis.setValue (SoRotationXYZ::X);

    //////////////////////////////////////
    // NPS AUV hull body center is at [0.0 0.0 0.0],
    // volumetric center of buoyancy

    // fin transformations forward

    SoTransform *xf1 = new SoTransform;
    xf1->translation.setValue( FINOFFSETFORWARD, 0.0, FINOFFSETUP);
    SoTransform *xf2 = new SoTransform;
    xf2->translation.setValue( 0.0, 0.0, FINOFFSETDOWN - FINOFFSETUP);

```

```

SoTransform *xf3 = new SoTransform;
xf3->translation.setValue( FINOFFSETFORWARD, FINOFFSETLEFT, 0.0);
SoTransform *xf4 = new SoTransform;
xf4->translation.setValue( 0.0, FINOFFSETRIGHT - FINOFFSETLEFT, 0.0);

// fin transformations aft

SoTransform *xf5 = new SoTransform;
xf5->translation.setValue( FINOFFSETAFT, 0.0, FINOFFSETUP);
SoTransform *xf6 = new SoTransform;
xf6->translation.setValue( 0.0 , 0.0, FINOFFSETDOWN - FINOFFSETUP);

SoTransform *xf7 = new SoTransform;
xf7->translation.setValue( FINOFFSETAFT, FINOFFSETLEFT, 0.0);
SoTransform *xf8 = new SoTransform;
xf8->translation.setValue( 0.0, FINOFFSETRIGHT - FINOFFSETLEFT, 0.0);

// 90 degree increment rotations - get #define'd PI values

SoRotationXYZ *ro90x = new SoRotationXYZ;
ro90x->angle.setValue (3.141592653 / 2.0);
ro90x-> axis.setValue (SoRotationXYZ::X);

SoRotationXYZ *ro90y = new SoRotationXYZ;
ro90y->angle.setValue (3.141592653 / 2.0);
ro90y-> axis.setValue (SoRotationXYZ::Y);

SoRotationXYZ *ro90z = new SoRotationXYZ;
ro90z->angle.setValue (3.141592653 / 2.0);
ro90z-> axis.setValue (SoRotationXYZ::Z);

SoRotationXYZ *ro180x = new SoRotationXYZ;
ro180x->angle.setValue (3.141592653);
ro180x-> axis.setValue (SoRotationXYZ::X);

SoRotationXYZ *ro180y = new SoRotationXYZ;
ro180y->angle.setValue (3.141592653);
ro180y-> axis.setValue (SoRotationXYZ::Y);

SoRotationXYZ *ro180z = new SoRotationXYZ;
ro180z->angle.setValue (3.141592653);
ro180z-> axis.setValue (SoRotationXYZ::Z);

SoRotationXYZ *ro270x = new SoRotationXYZ;
ro270x->angle.setValue (- 3.141592653 / 2.0);
ro270x-> axis.setValue (SoRotationXYZ::X);

SoRotationXYZ *ro270y = new SoRotationXYZ;
ro270y->angle.setValue (- 3.141592653 / 2.0);
ro270y-> axis.setValue (SoRotationXYZ::Y);

SoRotationXYZ *ro270z = new SoRotationXYZ;
ro270z->angle.setValue (- 3.141592653 / 2.0);
ro270z-> axis.setValue (SoRotationXYZ::Z);

// construct NPS AUV hull body
SoCube *hull = new SoCube;
hull->width = HULLBODYLENGTH;
hull->height = HULLBODYWIDTH;
hull->depth = HULLBODYHEIGHT;

// construct a control fin
SoCube *fin = new SoCube;
fin->width = FINLENGTH;
fin->height = FINWIDTH;
fin->depth = FINHEIGHT;

```

```

// construct fin pairs

rotate_AUV_bow_rudders = new SoRotationXYZ;
rotate_AUV_bow_rudders->axis.setValue (SoRotationXYZ::Z);
rotate_AUV_bow_rudders->angle.setValue ( 0.0 );

rotate_AUV_stern_rudders = new SoRotationXYZ;
rotate_AUV_stern_rudders->axis.setValue (SoRotationXYZ::Z);
rotate_AUV_stern_rudders->angle.setValue ( 0.0 );

rotate_AUV_bow_planes = new SoRotationXYZ;
rotate_AUV_bow_planes->axis.setValue (SoRotationXYZ::Z);
rotate_AUV_bow_planes->angle.setValue ( 0.0 );

rotate_AUV_stern_planes = new SoRotationXYZ;
rotate_AUV_stern_planes->axis.setValue (SoRotationXYZ::Z);
rotate_AUV_stern_planes->angle.setValue ( 0.0 );

// construct forward vertical fins (bow rudders)
SoSeparator *fvfins = new SoSeparator;
fvfins->addChild( xf1 );
fvfins->addChild( rotate_AUV_bow_rudders );
fvfins->addChild( fin );
fvfins->addChild( xf2 );
fvfins->addChild( ro180x ); // net rotation 180
fvfins->addChild( fin );

// construct aft vertical fins (stern rudders)
SoSeparator *avfins = new SoSeparator;
avfins->addChild( xf5 );
avfins->addChild( rotate_AUV_stern_rudders );
avfins->addChild( fin );
avfins->addChild( xf6 );
avfins->addChild( ro180x ); // net rotation 180
avfins->addChild( fin );

// construct forward horizontal fins (bow planes)
SoSeparator *fhfins = new SoSeparator;
fhfins->addChild( xf3 );
fhfins->addChild( ro90x ); // net rotation 90
fhfins->addChild( rotate_AUV_bow_planes );
fhfins->addChild( fin );
fhfins->addChild( ro270x );
fhfins->addChild( xf4 );
fhfins->addChild( ro270x ); // net rotation 270 (in case of fin asymmetry)
fhfins->addChild( fin );

// construct aft horizontal fins (stern planes)
SoSeparator *ahfins = new SoSeparator;
ahfins->addChild( xf7 );
ahfins->addChild( ro90x ); // net rotation 90
ahfins->addChild( rotate_AUV_stern_planes );
ahfins->addChild( fin );
ahfins->addChild( ro270x );
ahfins->addChild( xf8 );
ahfins->addChild( ro270x ); // net rotation 270 (in case of fin asymmetry)
ahfins->addChild( fin );

// construct cylinders to represent the thrusters
SoTransform *xf13 = new SoTransform;
xf13->translation.setValue(THRUSTERFORWARDV, 0.0, 0.0);
SoTransform *xf14 = new SoTransform;
xf14->translation.setValue(THRUSTERAFTV, 0.0, 0.0);
SoTransform *xf15 = new SoTransform;
xf15->translation.setValue(THRUSTERFORWARDH, 0.0, 0.0);
SoTransform *xf16 = new SoTransform;
xf16->translation.setValue(THRUSTERAFTH, 0.0, 0.0);

```

```

SoCylinder * tubeV = new SoCylinder;
tubeV->radius = THRUSTERID;
tubeV->height = HULLBODYHEIGHT + 0.2;
SoCylinder * tubeH = new SoCylinder;
tubeH->radius = THRUSTERID;
tubeH->height = HULLBODYWIDTH + 0.2;

// still inches
topsideBow-> translation.setValue( 0, TOPHALF + AUV_bow_vertical, 0);
bottomsideBow-> translation.setValue( 0, BOTTOMHALF + AUV_bow_vertical, 0);
bottomsideBow-> rotation.setValue(SbVec3f ( 0.0, 1.0, 0.0 ), M_PI );

topsideStern-> translation.setValue(0,TOPHALF + AUV_stern_vertical,0);
bottomsideStern-> translation.setValue(0,BOTTOMHALF + AUV_stern_vertical,0);
bottomsideStern-> rotation.setValue(SbVec3f ( 0.0, 1.0, 0.0), M_PI );

leftsideBow-> translation.setValue( 0, LEFT_HALF + AUV_bow_lateral, 0);
leftsideBow-> rotation.setValue(SbVec3f ( 0.0, 1.0, 0.0 ), M_PI );
rightsideBow-> translation.setValue( 0, RIGHTHALF + AUV_bow_lateral, 0);

leftsideStern-> translation.setValue(0, LEFT_HALF + AUV_stern_lateral, 0);
leftsideStern-> rotation.setValue(SbVec3f ( 0.0, 1.0, 0.0 ), M_PI );
rightsideStern-> translation.setValue(0, RIGHTHALF + AUV_stern_lateral, 0);

thrusterWakeFV = new SoCone; // global for callbacks
thrusterWakeFV->height = AUV_bow_vertical * 2.0;
thrusterWakeFV->bottomRadius = AUV_bow_vertical / 4.0;
thrusterWakeFV->parts = SoCone::SIDES;

SoSeparator * thrusterFV = new SoSeparator;
thrusterFV->addChild( xf13 );
thrusterFV->addChild( ro90x );
thrusterFV->addChild( tubeV );
thrusterFV->addChild( wires );
thrusterFV->addChild( wakeComplexity );
thrusterFV->addChild( seagreen );
thrusterFV->addChild( whichWakeFV );

thrusterFV->addChild( thrusterWakeFV );

thrusterWakeAV = new SoCone; // global for callbacks
thrusterWakeAV->height = AUV_stern_vertical * 2.0;
thrusterWakeAV->bottomRadius = AUV_stern_vertical / 4.0;
thrusterWakeAV->parts = SoCone::SIDES;

SoSeparator * thrusterAV = new SoSeparator;
thrusterAV->addChild( xf14 );
thrusterAV->addChild( ro90x );
thrusterAV->addChild( tubeV );
thrusterAV->addChild( wires );
thrusterAV->addChild( wakeComplexity );
thrusterAV->addChild( seagreen );
thrusterAV->addChild( whichWakeAV );
thrusterAV->addChild( thrusterWakeAV );

thrusterWakeFH = new SoCone; // global for callbacks
thrusterWakeFH->height = AUV_bow_lateral * 2.0;
thrusterWakeFH->bottomRadius = AUV_bow_lateral / 4.0;
thrusterWakeFH->parts = SoCone::SIDES;

SoSeparator * thrusterFH = new SoSeparator;
thrusterFH->addChild( xf15 );
thrusterFH->addChild( ro90y );
thrusterFH->addChild( tubeH );
thrusterFH->addChild( ro180z );
thrusterFH->addChild( wires );
thrusterFH->addChild( wakeComplexity );
thrusterFH->addChild( seagreen );

```

```

thrusterFH->addChild( whichWakeFH );
thrusterFH->addChild( thrusterWakeFH );

thrusterWakeAH = new SoCone; // global for callbacks
thrusterWakeAH->height      = AUV_stern_lateral * 2.0;
thrusterWakeAH->bottomRadius = AUV_stern_lateral / 4.0;
thrusterWakeAH->parts       = SoCone::SIDES;

SoSeparator * thrusterAH = new SoSeparator;
thrusterAH->addChild( xf16 );
thrusterAH->addChild( ro90y );
thrusterAH->addChild( tubeH );
thrusterAH->addChild( ro180z );
thrusterAH->addChild( wires );
thrusterAH->addChild( wakeComplexity );
thrusterAH->addChild( seagreen );
thrusterAH->addChild( whichWakeAH );
thrusterAH->addChild( thrusterWakeAH );

// construct internal CARDCAGES left and right

SoCube *cardcageleftbox = new SoCube;
cardcageleftbox->width  = CARDCAGELENGTH;
cardcageleftbox->height = CARDCAGEWIDTH;
cardcageleftbox->depth  = CARDCAGEHEIGHT;

SoTransform *xfcardcageleft = new SoTransform;
xfcardcageleft->translation.setValue( 0.0, CARDCAGELEFT, 0.0 );

SoSeparator *cardcageleft = new SoSeparator;
cardcageleft->addChild( xfcardcageleft );
cardcageleft->addChild( cardcageleftbox );

SoCube *cardcagerightbox = new SoCube;
cardcagerightbox->width  = CARDCAGELENGTH;
cardcagerightbox->height = CARDCAGEWIDTH;
cardcagerightbox->depth  = CARDCAGEHEIGHT;

SoTransform *xfcardcageright = new SoTransform;
xfcardcageright->translation.setValue( 0.0, CARDCAGERIGHT, 0.0 );

SoSeparator *cardcageright = new SoSeparator;
cardcageright->addChild( xfcardcageright );
cardcageright->addChild( cardcagerightbox );

////////////////////////////////////
// construct main body out of parts
SoGroup *body = new SoGroup;

body->addChild( gold );
body->addChild( fvfins );
body->addChild( avfins );
body->addChild( fhfins );
body->addChild( ahfins );

body->addChild( npsblue );
body->addChild( hull );

body->addChild( silver );
body->addChild( thrusterFV );
body->addChild( thrusterAV );
body->addChild( thrusterFH );
body->addChild( thrusterAH );

body->addChild( cardcageleft );
body->addChild( cardcageright );

```



```

////////////////////////////////////
// construct nosecone using a NURBS surface

```

```

static float pts [25][3] =
{
    { - SEAM,                LEFT,      TOP      },
    { - SEAM,                LEFT_HALF, TOP      },
    { - SEAM,                CENTER,    TOP      },
    { - SEAM,                RIGHTHALF, TOP      },
    { - SEAM,                RIGHT,     TOP      },

    { - SEAM,                LEFT,      TOPHALF },
    { - SEAM + DOMECONTROLPTHALF, LEFT_HALF, TOPHALF },
    { - SEAM + DOMECONTROLPTHALF, CENTER,  TOPHALF },
    { - SEAM + DOMECONTROLPTHALF, RIGHTHALF, TOPHALF },
    { - SEAM,                RIGHT,     TOPHALF },

    { - SEAM,                LEFT,      CENTER },
    { - SEAM + DOMECONTROLPTHALF, LEFT_HALF, CENTER },
    { - SEAM + DOMECONTROLPT,    CENTER,  CENTER },
    { - SEAM + DOMECONTROLPTHALF, RIGHTHALF, CENTER },
    { - SEAM,                RIGHT,     CENTER },

    { - SEAM,                LEFT,      BOTTOMHALF },
    { - SEAM + DOMECONTROLPTHALF, LEFT_HALF, BOTTOMHALF },
    { - SEAM + DOMECONTROLPTHALF, CENTER,  BOTTOMHALF },
    { - SEAM + DOMECONTROLPTHALF, RIGHTHALF, BOTTOMHALF },
    { - SEAM,                RIGHT,     BOTTOMHALF },

    { - SEAM,                LEFT,      BOTTOM },
    { - SEAM,                LEFT_HALF, BOTTOM },
    { - SEAM,                CENTER,    BOTTOM },
    { - SEAM,                RIGHTHALF, BOTTOM },
    { - SEAM,                RIGHT,     BOTTOM },
};

static float knots [10] = .
    ( 0, 0, 0, 0, 0, 1, 1, 1, 1, 1 );

SoComplexity *noseconeComplexity = new SoComplexity;
noseconeComplexity->value = 0.7;

SoCoordinate3 *controlPts = new SoCoordinate3;
controlPts->point.setValues ( 0, 25, pts );

SoNurbsSurface *sonardome = new SoNurbsSurface;
sonardome->numUControlPoints.setValue ( 5 );
sonardome->numVControlPoints.setValue ( 5 );
sonardome->uKnotVector.setValues ( 0, 10, knots );
sonardome->vKnotVector.setValues ( 0, 10, knots );

SoSeparator *nosesection = new SoSeparator;
nosesection->addChild( npsblue );
nosesection->addChild( noseconeComplexity );
nosesection->addChild( controlPts );
nosesection->addChild( sonardome );

// Initialize the Cone for Representing the ST1000 Beam
coneSonarST1000 = new SoCone; // global for callbacks
if (AUV_ST1000_range > 0.0)
{
    coneSonarST1000->height = fabs(AUV_ST1000_range);
    coneSonarST1000->bottomRadius = fabs(AUV_ST1000_range)/60.0; // 1 degree
    sonar_cone_color1000->ambientColor.setValue ( 1.0, 0.0, 0.15 );
    sonar_cone_color1000->diffuseColor.setValue ( 1.0, 0.0, 0.15 );
    sonar_cone_color1000->specularColor.setValue ( 1.0, 0.0, 0.15 );
    ST1000Complexity->value.setValue ( ST1000Complexity_on );
}

```

```

    }
    else if (AUV_ST1000_range < 0.0)
    {
        coneSonarST1000->height = 0.001;
        coneSonarST1000->bottomRadius = 0.0001;
        xfConeSonarST1000->translation.setValue ( 0.0, 0.0, 0.0 );
        sonar_cone_color1000->ambientColor.setValue ( 0.0, 0.0, 0.0 );
        sonar_cone_color1000->diffuseColor.setValue ( 0.7, 0.7, 0.0 );
        sonar_cone_color1000->specularColor.setValue ( 0.7, 0.7, 0.0 );
        ST1000Complexity->value.setValue ( ST1000Complexity_off );
    }
    else // zero range, draw tiny initialization no-detection cone
    {
        coneSonarST1000->height = 0.001;
        coneSonarST1000->bottomRadius = 0.0001;
        sonar_cone_color1000->ambientColor.setValue ( 0.0, 0.0, 0.0 );
        sonar_cone_color1000->diffuseColor.setValue ( 0.7, 0.7, 0.0 );
        sonar_cone_color1000->specularColor.setValue ( 0.7, 0.7, 0.0 );
        ST1000Complexity->value.setValue ( ST1000Complexity_off );
    }
    coneSonarST1000->parts = SoCone::SIDES;

    SoSeparator * sepSonar1000 = new SoSeparator;

    // drawn from center
    if (AUV_ST1000_range > 0.0)
        xfConeSonarST1000->translation.setValue( 0.0,
                                                    - (AUV_ST1000_range / 2.0),
                                                    0.0 );
    else xfConeSonarST1000->translation.setValue ( 0.0, 0.0, 0.0 );

    if (TRACE) {
        cout << "xfConeSonarST1000->translation.x = " << xfConeSonarST1000-
>translation.getValue()[0] << endl;
        cout << "xfConeSonarST1000->translation.y = " << xfConeSonarST1000-
>translation.getValue()[1] << endl;
        cout << "xfConeSonarST1000->translation.z = " << xfConeSonarST1000-
>translation.getValue()[2] << endl;
    }
    offsetST1000->translation.setValue( AUV_ST1000_x_offset / 12.0,
                                         -AUV_ST1000_y_offset / 12.0,
                                         -AUV_ST1000_z_offset / 12.0 );

    if (TRACE) {
        cout << "offsetST1000->translation.x = "
        << offsetST1000->translation.getValue()[0] << endl;
        cout << "offsetST1000->translation.y = "
        << offsetST1000->translation.getValue()[1] << endl;
        cout << "offsetST1000->translation.z = "
        << offsetST1000->translation.getValue()[2] << endl;
    }
    sepSonar1000->addChild( offsetST1000 );
    sepSonar1000->addChild( ro90z );
    rotConeSonarST1000->axis = SoRotationXYZ::Z;
    sepSonar1000->addChild( rotConeSonarST1000 );
    sepSonar1000->addChild( wires );
    sepSonar1000->addChild( ST1000Complexity );
    sepSonar1000->addChild( sonar_cone_color1000 );
    sepSonar1000->addChild( xfConeSonarST1000 );
    sepSonar1000->addChild( coneSonarST1000 );

    /* Initialize the Cone for Representing the ST725 Beam */
    coneSonarST725 = new SoCone; // global for callbacks
    if (AUV_ST725_range > 0.0)
    {
        coneSonarST725->height = fabs (AUV_ST725_range);
        coneSonarST725->bottomRadius = fabs (AUV_ST725_range) / 60.0; // 1 degree
        sonar_cone_color725->ambientColor.setValue ( 0.8, 0.0, 0.8 );
    }

```

```

        sonar_cone_color725->diffuseColor.setValue ( 0.8, 0.0, 0.8 );
        sonar_cone_color725->specularColor.setValue ( 0.8, 0.0, 0.8 );
        ST725Complexity->value.setValue ( ST725Complexity_on );
    }
    else if (AUV_ST725_range < 0.0)
    {
        coneSonarST725->height = 0.001;
        coneSonarST725->bottomRadius = 0.0001;
        xfConeSonarST725->translation.setValue ( 0.0, 0.0, 0.0 );
        sonar_cone_color725->ambientColor.setValue ( 0.0, 0.0, 0.0 );
        sonar_cone_color725->diffuseColor.setValue ( 0.0, 0.0, 0.0 );
        sonar_cone_color725->specularColor.setValue ( 0.7, 0.7, 0.7 );
        ST725Complexity->value.setValue ( ST725Complexity_off );
    }
    else // zero range, draw tiny initialization no-detection cone
    {
        coneSonarST725->height = 0.001;
        coneSonarST725->bottomRadius = 0.0001;
        sonar_cone_color725->ambientColor.setValue ( 0.0, 0.0, 0.0 );
        sonar_cone_color725->diffuseColor.setValue ( 0.0, 0.0, 0.0 );
        sonar_cone_color725->specularColor.setValue ( 0.7, 0.7, 0.7 );
        ST725Complexity->value.setValue ( ST725Complexity_off );
    }
    coneSonarST725->parts = SoCone::SIDES;

    // drawn from center
    if (AUV_ST725_range > 0.0)
        xfConeSonarST725->translation.setValue( 0.0,
                                                - (AUV_ST725_range/ 2.0),
                                                0.0 );
    else xfConeSonarST725->translation.setValue ( 0.0, 0.0, 0.0 );

    SoSeparator * sepSonar725 = new SoSeparator;

    offsetST725->translation.setValue( AUV_ST725_x_offset / 12.0,
                                       -AUV_ST725_y_offset / 12.0,
                                       -AUV_ST725_z_offset / 12.0 );
    sepSonar725->addChild( offsetST725 );
    sepSonar725->addChild( ro90z );
    rotConeSonarST725->axis = SoRotationXYZ::Z;
    sepSonar725->addChild( rotConeSonarST725 );
    scaleConeSonarST725->scaleFactor.setValue ( 1.0, 1.0, 12.0 );
    sepSonar725->addChild( scaleConeSonarST725 );
    sepSonar725->addChild( wires );
    sepSonar725->addChild( ST725Complexity );
    sepSonar725->addChild( sonar_cone_color725 );
    sepSonar725->addChild( xfConeSonarST725 );
    sepSonar725->addChild( coneSonarST725 );

    ////////////////////////////////////////
    // Define tail section

    // ensure polygons are defined in clockwise fashion!

    // Two triangles using SoTriangleStripSet:
    static int32_t numbertrianglevertices [2] = {3, 3};
    // static long numbertrianglevertices [2] = {3, 3};
    // type changed from long to int32_t for Inventor 2.1 upgrade

    static float afttrianglevertices [6][3] =
    {
        {STERN, LEFT, 0.0}, {SEAM, LEFT, TOP}, {SEAM, LEFT, BOTTOM},
        {STERN, RIGHT, 0.0}, {SEAM, RIGHT, BOTTOM}, {SEAM, RIGHT, TOP},
    };

    // Define coordinates for triangular vertices & SoTriangleStripSet
    SoCoordinate3 *tailcoord1 = new SoCoordinate3;
    tailcoord1->point.setValues (0, 6, afttrianglevertices);

```

```

SoTriangleStripSet *tailtrianglestripset = new SoTriangleStripSet;
tailtrianglestripset->numVertices.setValues (0, 2, numbertrianglestripvertices);

// Two rectangles using FaceSet:
static int32_t numberquadvertices [2] = {4, 4}; // ref. p. 5-6
// static long    numberquadvertices [2] = {4, 4}; // ref. p. 5-6
// type changed from long to int32_t for Inventor 2.1 upgrade

static float aftquadvertices [8][3] =
{
    {STERN, LEFT, 0.0}, {STERN, RIGHT, 0.0}, {SEAM, RIGHT, TOP},
    {SEAM, LEFT, TOP},

    {STERN, RIGHT, 0.0}, {STERN, LEFT, 0.0}, {SEAM, LEFT, BOTTOM},
    {SEAM, RIGHT, BOTTOM},
};

// Define coordinates for quad vertices & SoFaceSet
SoCoordinate3 *tailcoord2 = new SoCoordinate3;
tailcoord2->point.setValues (0, 8, aftquadvertices);

SoFaceSet *tailquadset = new SoFaceSet;

tailquadset->numVertices.setValues (0, 2, numberquadvertices);

// Two cylinders currently represent the propellers - improve this!
// a much fancier individual prop model possible here; also add complexity
SoCylinder *prop = new SoCylinder;
prop->radius = 2.00;
prop->height = 1.00;

// Two cylinders to represent the shafts
SoCylinder *shaft = new SoCylinder;
shaft->radius = 0.50;
shaft->height = 4.00;

SoTransform *xf18 = new SoTransform; // shafts relative to props
// note: rotated 90z
xf18->translation.setValue(0.0, -2.0, 0.0);

SoTransform *xf11 = new SoTransform; // left prop
xf11->translation.setValue(STERN - 2.0, SHAFTOFFSETLEFT, 0.0);

// compose shaft with prop
SoSeparator *leftprop = new SoSeparator;
leftprop->addChild( xf11 );
leftprop->addChild( ro90z );
leftprop->addChild( prop );
leftprop->addChild( xf18 );
leftprop->addChild( shaft );

SoTransform *xfPropellerWakePort = new SoTransform;
xfPropellerWakePort->translation.setValue( 0.0, 15.0, 0.0 );
SoSeparator * separatorPropellerWakePort = new SoSeparator;
separatorPropellerWakePort->addChild( xfPropellerWakePort );
separatorPropellerWakePort->addChild( ro180x );
separatorPropellerWakePort->addChild( wires );
separatorPropellerWakePort->addChild( wakeComplexity );
separatorPropellerWakePort->addChild( seagreen );
conePropellerWakePort = new SoCone; // global for callbacks
conePropellerWakePort->height = AUV_port_rpm / 100.0 * 24.0;
conePropellerWakePort->bottomRadius = fabs (AUV_port_rpm) / 100.0 * 6.0;
conePropellerWakePort->parts = SoCone::SIDES;
separatorPropellerWakePort->addChild( conePropellerWakePort );
leftprop->addChild( separatorPropellerWakePort );

SoTransform *xf12 = new SoTransform; // right props

```

```

xf12->translation.setValue(STERN - 2.0, SHAFTOFFSETRIGHT, 0.0);

SoSeparator *rightprop = new SoSeparator;
rightprop->addChild( xf12 );
rightprop->addChild( ro90z );
rightprop->addChild( prop );
rightprop->addChild( xf18 );
rightprop->addChild( shaft );

SoTransform *xfPropellerWakeStbd = new SoTransform;
xfPropellerWakeStbd->translation.setValue( 0.0, 15.0, 0.0 );
SoSeparator * separatorPropellerWakeStbd = new SoSeparator;
separatorPropellerWakeStbd->addChild( xfPropellerWakeStbd );
separatorPropellerWakeStbd->addChild( ro180x );
separatorPropellerWakeStbd->addChild( wires );
separatorPropellerWakeStbd->addChild( wakeComplexity );
separatorPropellerWakeStbd->addChild( seagreen );
conePropellerWakeStbd = new SoCone; // global for callbacks
conePropellerWakeStbd->height = AUV_port_rpm / 100.0 * 24.0;
conePropellerWakeStbd->bottomRadius = fabs( AUV_port_rpm ) / 100.0 * 6.0;
conePropellerWakeStbd->parts = SoCone::SIDES;
separatorPropellerWakeStbd->addChild( conePropellerWakeStbd );
rightprop->addChild( separatorPropellerWakeStbd );

// construct stern box support beneath aft vertical fins
SoCube *sternfinsupportcube = new SoCube;
sternfinsupportcube->width = FINLENGTH + 1.5;
sternfinsupportcube->height = 5.0;
sternfinsupportcube->depth = HULLBODYHEIGHT - 0.5;

SoTransform *xf17 = new SoTransform;
xf17->translation.setValue(FINOFFSETAFT, 0.0, 0.0);

SoSeparator *sternfinsupport = new SoSeparator;
sternfinsupport->addChild( xf17 );
sternfinsupport->addChild( sternfinsupportcube );

SoSeparator *tailsection = new SoSeparator;
tailsection->addChild( npsblue );
tailsection->addChild( tailcoord1 );
tailsection->addChild( tailtriangleset );
tailsection->addChild( tailcoord2 );
tailsection->addChild( tailquadset );
tailsection->addChild( sternfinsupport );
tailsection->addChild( brass );
tailsection->addChild( leftprop );
tailsection->addChild( rightprop );

////////////////////////////////////
// robot is just units & body & nosesction & tailsection & extra stuff

SoSeparator * robot = new SoSeparator;

// ivview/ivquicken fails on SoUnits nodes so SoScale nodes used originally
// robot->addChild( unitsfeet );
// robot->addChild( MetersToFeet );

// robot root transform for overall vehicle orientation
AUV_position_node = new SoTransform;
AUV_position_node->translation.setValue(0.0, 0.0, 0.0);
robot->addChild( AUV_position_node );

rotate_AUV_z = new SoRotationXYZ;
rotate_AUV_z->angle.setValue ( - AUV_psi);
rotate_AUV_z->axis.setValue (SoRotationXYZ::Z);
robot->addChild( rotate_AUV_z );

rotate_AUV_y = new SoRotationXYZ;

```

```

rotate_AUV_y->angle.setValue ( - AUV_theta);
rotate_AUV_y->axis.setValue (SoRotationXYZ::Y);
robot->addChild( rotate_AUV_y );

rotate_AUV_x = new SoRotationXYZ;
rotate_AUV_x->angle.setValue ( AUV_phi);
rotate_AUV_x->axis.setValue (SoRotationXYZ::X);
robot->addChild( rotate_AUV_x );

robot->addChild( sepSonar725 ); // feet
robot->addChild( sepSonar1000 );

// ivview/ivquicken fails on SoUnits nodes so SoScale nodes used originally
SoUnits *unitsinches = new SoUnits;
unitsinches->units.setValue ( SoUnits::INCHES );
robot->addChild( unitsinches );
// robot->addChild( FeetToInches );

SoPickStyle *unpickablestylenode;
SoPickStyle *pickablestylenode;
unpickablestylenode = new SoPickStyle;
pickablestylenode = new SoPickStyle;
unpickablestylenode->style.setValue ( SoPickStyle::UNPICKABLE );
pickablestylenode->style.setValue ( SoPickStyle::SHAPE );

// Make subsequent nodes unpickable so that AUV is treated as a whole
robot->addChild( unpickablestylenode );

robot->addChild( body );
robot->addChild( nosessection );
robot->addChild( tailsection );
/*
SoTransform *xf19 = new SoTransform;
xf19->translation.setValue( 0.0, 0.0, - 2.0);
robot->addChild( xf19 );
robot->addChild( new SoPointLight );
cout << "new point light added under robot" << endl;
*/
return robot;
}

////////////////////////////////////

double sign (double x)
{
    if (x > 0.0) return 1.0;
    else if (x < 0.0) return -1.0;
    else return 1.0;
}

//-----//

double degrees (double x) // radians input
{
    return x * 180.0 / PI;
}

//-----//

double radians (double x) // degrees input
{
    return x * PI / 180.0;
}

//-----//

double arclamp (double x)

```

```

{
    if      (x >  1.0)
    {
        x = 1.0;
        cout << "viewer: arcclamp reduced " << x << " to 1.0" << endl;
    }
    else if (x < -1.0)
    {
        x = -1.0;
        cout << "viewer: arcclamp raised " << x << " to -1.0" << endl;
    }
    return x;
}
//-----//

double dnormalize (double angle_radians)
{
    double new_angle = angle_radians;

    while (new_angle > 2*PI) new_angle -= 2*PI;
    while (new_angle < 0.0 ) new_angle += 2*PI;
    return new_angle;
}
//-----//
-//

void swap_float (float a, float b)
{
    float temp;
    temp = b;
    b = a;
    a = temp;
}
//-----//

static int DIS_net_open () // Ref: macedonia include files
{
    // Multicast Defaults from
    // /n/elsie/work3/macedoni/net/mcast/network/utills/planes/planes.cc

    //      ttl value does not matter since this viewing program only reads PDUs
    u_char ttl          = 1;          // multicast ttl= 1 stays inside LAN
                                   // multicast ttl= 15 stays inside NPS
                                   // multicast ttl=127 is global

    int      exercise_id      = -1;
    int      coordinate_system = 0; // 0 = flat world, 1 = round world
    char * utm_file           = "";

    int result = net_open (port, group, ttl,
                           exercise_id, coordinate_system, utm_file);

// int result = net_open (port, group, ttl); // old version

    if (result == FALSE)
    {
        cout << "viewer: DIS_net_open () failed" << endl;
        DIS_port_open = FALSE;
    }
    else
    {
        DIS_port_open = TRUE;

        // cout << "viewer: port = " << port << ", group = " << group
        //      << ", ttl = " << ttl << endl;
        // cout << " exercise_id = " << exercise_id

```

```

        //      << " , coordinate_system = "      << coordinate_system
        //      << " , utm_file = \"\" << utm_file << "\"\" << endl;
    }
    return result;
}
//-----//

static void DIS_Redraw_Callback ( void      * unused_data,
                                SoSensor * unused_calling_sensor )
{
    unused_data      = unused_data;
    unused_calling_sensor = unused_calling_sensor; // eliminate warnings

    double delta_x    = 0.0;
    double delta_y    = 0.0;
    double delta_z    = 0.0;
    double delta_phi   = 0.0;
    double delta_theta = 0.0;
    double delta_psi   = 0.0;

    int                number_of_PDUs;

    // EntityID                UUV_DIS_id;

    // EntityType                UUV_DIS_type;

    EntityStatePDU        * UUV_DIS_pdu = NULL;

    char                    * local_PDU   = NULL;

    PDUType                local_PDU_type;

    // static int rcvd = 0;

    char    NPSAUV_Marking [11];
    bzero   (NPSAUV_Marking, 11);
    strncpy (NPSAUV_Marking, "Phoenix AUV", 11); // 11 chars max

    if ((delta_time <= 5.5) && (delta_time >= 0.0) && (DEADRECKON == TRUE))
    {
        // cout << "viewer: DIS_Redraw_Callback: PDU loop delta_time = "
        //      << delta_time << endl;
    }

    while (TRUE) // until break
    {
        number_of_PDUs = net_read (& local_PDU, & local_PDU_type); // old version
//      number_of_PDUs = net_read (& local_PDU, & local_PDU_type, & inaddr);

        // cout << "viewer: net_read complete, number_of_PDUs available = "
        //      << number_of_PDUs << endl;

        if (number_of_PDUs <= -1)
            cout << "viewer: Error on net_read, number_of_PDUs = "
                << number_of_PDUs << endl;

        else if (number_of_PDUs <= 0)
        {
            break; // no more PDUs, done for now
        }

        // rcvd ++;
        // cout << "PDU received, rcvd = " << rcvd << endl;
        // printPDU (local_PDU);

        UUV_DIS_pdu = (EntityStatePDU *) local_PDU;

        // ensure the PDU values are the right types

```



```

if (local_PDU_type != EntityStatePDU_Type)
{
    cout << "viewer: local_PDU_type != EntityStatePDU_Type, ignored..."
        << endl;
    printPDU (local_PDU);
    // don't forget freePDU or get a memory leak!
    // articulated parameters are also freed
    freePDU ((char *) UUV_DIS_pdu);
    cout << "viewer: freePDU ((char *) UUV_DIS_pdu) called for this PDU"
        << endl;

    continue; // not a break since other PDUs may be waiting
}
else if (strncmp ((char *) UUV_DIS_pdu->entity_marking.markings,
                  NPSAUV_Marking, 11) != 0)
{
    cout << "viewer: non-NPS AUV Entity State PDU encountered, ignored"
        << endl;
    printPDU (local_PDU);
    freePDU ((char *) UUV_DIS_pdu);
    // don't forget freePDU or get a memory leak!
    // articulated parameters are also freed
    cout << "viewer: freePDU ((char *) UUV_DIS_pdu) called for this PDU"
        << endl;

    continue; // not a break since other PDUs may be waiting
}
// cout << "PDU OK" << endl;

// extract parameters of an entity state PDU (most are listed in pdu.h)
// this assumes there are no articulated parameters (add later) <<<

// DIS ID and Type
// UUV_DIS_id = UUV_DIS_pdu->entity_id;
// UUV_DIS_type = UUV_DIS_pdu->entity_type;

time_stamp_of_current_PDU = UUV_DIS_pdu->entity_state_header.time_stamp;
time_of_PDU_receipt = clock ();

PDU_overdue = FALSE;

// Posture
AUV_x = UUV_DIS_pdu->entity_location.x * FT_PER_METERS;
AUV_y = UUV_DIS_pdu->entity_location.y * FT_PER_METERS;
AUV_z = UUV_DIS_pdu->entity_location.z * FT_PER_METERS;
AUV_phi = radians (UUV_DIS_pdu->entity_orientation.phi);
AUV_theta = radians (UUV_DIS_pdu->entity_orientation.theta);
AUV_psi = radians (UUV_DIS_pdu->entity_orientation.psi);

// cout << endl;
// cout << "viewer: DIS_net_read posture trace:" << endl;
cout << "[";
cout << UUV_DIS_pdu->entity_location.x << ", ";
cout << UUV_DIS_pdu->entity_location.y << ", ";
cout << UUV_DIS_pdu->entity_location.z << ", ";
cout << UUV_DIS_pdu->entity_orientation.phi << ", ";
cout << UUV_DIS_pdu->entity_orientation.theta << ", ";
cout << UUV_DIS_pdu->entity_orientation.psi << "]" << endl;

// Linear & angular velocities in body coordinates/meters by DIS standard
AUV_x_dot = UUV_DIS_pdu->entity_velocity.x * FT_PER_METERS;
AUV_y_dot = UUV_DIS_pdu->entity_velocity.y * FT_PER_METERS;
AUV_z_dot = UUV_DIS_pdu->entity_velocity.z * FT_PER_METERS;

AUV_phi_dot =
    radians(UUV_DIS_pdu->dead_reckon_params.angular_velocity[0]);

```

```

AUV_theta_dot=
    radians(UUV_DIS_pdu->dead_reckon_params.angular_velocity[1]);

AUV_psi_dot =
    radians(UUV_DIS_pdu->dead_reckon_params.angular_velocity[2]);

//      cout << "viewer: World coordinate velocities:  ";
//      cout << "[";
//      cout <<  UUV_DIS_pdu->entity_velocity.x          << ", ";
//      cout <<  UUV_DIS_pdu->entity_velocity.y          << ", ";
//      cout <<  UUV_DIS_pdu->entity_velocity.z          << ", ";
//      cout <<  UUV_DIS_pdu->dead_reckon_params.angular_velocity [0] << ", ";
//      cout <<  UUV_DIS_pdu->dead_reckon_params.angular_velocity [1] << ", ";
//      cout <<  UUV_DIS_pdu->dead_reckon_params.angular_velocity [2] << "]"
//      << endl;
//      cout << endl;

// Note even though the accelerations are calculated in the superclass
// UUVBody, use of global state vector lets us construct class hierarchy
// based on problem structure instead of the communications dependencies.
// This is proposed as a general DIS-compatible vehicle object hierarchy.

// Accelerations not produced in world coordinates, thus zeroes expected
AUV_u_dot =
    UUV_DIS_pdu->dead_reckon_params.linear_accel [0] * FT_PER_METERS;
AUV_v_dot =
    UUV_DIS_pdu->dead_reckon_params.linear_accel [1] * FT_PER_METERS;
AUV_w_dot =
    UUV_DIS_pdu->dead_reckon_params.linear_accel [2] * FT_PER_METERS;

//      cout << "viewer: World coordinate accelerations: ";
//      cout << "[";
//      cout <<  UUV_DIS_pdu->dead_reckon_params.linear_accel [0] << ", ";
//      cout <<  UUV_DIS_pdu->dead_reckon_params.linear_accel [1] << ", ";
//      cout <<  UUV_DIS_pdu->dead_reckon_params.linear_accel [2] << "]" <<
endl;

// what we look like
// UUV_DIS_pdu->entity_appearance;
// UUV_DIS_pdu->entity_marking.character_set;
// UUV_DIS_pdu->entity_marking.markings;

// project our movement
UUV_DIS_pdu->dead_reckon_params.algorithm = DRAlgo_DRM_FPW;

// cout << "viewer: DIS_net_read () successful" << endl;
// cout << flush;

// update overall AUV posture (both position & orientation)
// account for shift to SGI/OpenInventor coordinate system
AUV_position_node->translation.setValue ( AUV_x, - AUV_y, - AUV_z);
rotate_AUV_z->          angle.setValue ( - AUV_psi);
rotate_AUV_y->          angle.setValue ( - AUV_theta);
rotate_AUV_x->          angle.setValue (  AUV_phi);

ArticulatParamsNode * APNptr = UUV_DIS_pdu->articulat_params_head;

AUV_time          =  APNptr->articulat_params.parameter_value [0];
AUV_time          +=  APNptr->articulat_params.parameter_value [1] / 10.0;
AUV_delta_rudder  =  APNptr->articulat_params.parameter_value [2];
AUV_delta_planes  =  APNptr->articulat_params.parameter_value [3];

// denormalize former shorts to be negative if necessary
if (AUV_delta_rudder >= 128.0) AUV_delta_rudder -= 256.0;
if (AUV_delta_planes >= 128.0) AUV_delta_planes -= 256.0;

AUV_port_rpm      =  APNptr->articulat_params.parameter_value [4];
if (AUV_port_rpm >= 128.0) AUV_port_rpm = AUV_port_rpm - 256.0;

```

```

AUV_port_rpm      *= 10.0;
if (AUV_port_rpm >= 0.0)
    AUV_port_rpm += APNptr->articulat_params.parameter_value [5];
else AUV_port_rpm -= APNptr->articulat_params.parameter_value [5];

AUV_stbd_rpm      = APNptr->articulat_params.parameter_value [6];
if (AUV_stbd_rpm >= 128.0) AUV_stbd_rpm = AUV_stbd_rpm - 256.0;
AUV_stbd_rpm      *= 10.0;
if (AUV_stbd_rpm >= 0.0)
    AUV_stbd_rpm += APNptr->articulat_params.parameter_value [7];
else AUV_stbd_rpm -= APNptr->articulat_params.parameter_value [7];

cout << "(" << AUV_time << ")" << endl;

// cout << "viewer: Articulation parameter 0:" << endl;
// cout << "AUV_time      = " << AUV_time      << endl;
// cout << "AUV_delta_rudder = " << AUV_delta_rudder << endl;
// cout << "AUV_delta_planes = " << AUV_delta_planes << endl;
// cout << "AUV_port_rpm    = " << AUV_port_rpm    << endl;
// cout << "AUV_stbd_rpm    = " << AUV_stbd_rpm    << endl;

APNptr = APNptr->next_articulat_params; // next articulated parameter

// thrusters
AUV_bow_vertical   = APNptr->articulat_params.parameter_value [0];
AUV_stern_vertical = APNptr->articulat_params.parameter_value [1];
AUV_bow_lateral    = APNptr->articulat_params.parameter_value [2];
AUV_stern_lateral  = APNptr->articulat_params.parameter_value [3];

// denormalize former shorts to be negative if necessary
if (AUV_bow_vertical   >= 128.0) AUV_bow_vertical   -= 256.0;
if (AUV_stern_vertical >= 128.0) AUV_stern_vertical -= 256.0;
if (AUV_bow_lateral    >= 128.0) AUV_bow_lateral    -= 256.0;
if (AUV_stern_lateral  >= 128.0) AUV_stern_lateral  -= 256.0;

// convert thruster volts to force by signed squares & normalize adjust
AUV_bow_vertical   = AUV_bow_vertical   * fabs(AUV_bow_vertical   ) /24.0;
AUV_stern_vertical = AUV_stern_vertical * fabs(AUV_stern_vertical) /24.0;
AUV_bow_lateral    = AUV_bow_lateral    * fabs(AUV_bow_lateral    ) /24.0;
AUV_stern_lateral  = AUV_stern_lateral  * fabs(AUV_stern_lateral  ) /24.0;

// slots 4..7 as yet unused

// cout << "viewer: Articulation parameter 1:  thrusters"
// cout << endl;
// cout << "AUV_bow_vertical   = " << AUV_bow_vertical   << endl;
// cout << "AUV_stern_vertical = " << AUV_stern_vertical << endl;
// cout << "AUV_bow_lateral    = " << AUV_bow_lateral    << endl;
// cout << "AUV_stern_lateral  = " << AUV_stern_lateral  << endl;

APNptr = APNptr->next_articulat_params; // next articulated parameter

AUV_ST1000_bearing = APNptr->articulat_params.parameter_value [0] * 10 +
    APNptr->articulat_params.parameter_value [1];
AUV_ST1000_range   = APNptr->articulat_params.parameter_value [2] / 4.0;
AUV_ST1000_strength = APNptr->articulat_params.parameter_value [3];
AUV_ST725_bearing  = APNptr->articulat_params.parameter_value [4] * 10 +
    APNptr->articulat_params.parameter_value [5];
AUV_ST725_range    = APNptr->articulat_params.parameter_value [6] / 4.0;
AUV_ST725_strength = APNptr->articulat_params.parameter_value [7];

// cout << "viewer: Articulation parameter 2:  sonar" << endl;
// cout << "AUV_ST1000_bearing = " << AUV_ST1000_bearing << endl;
// cout << "AUV_ST1000_range   = " << AUV_ST1000_range   << endl;
// cout << "AUV_ST1000_strength = " << AUV_ST1000_strength << endl;
// cout << "AUV_ST725_bearing  = " << AUV_ST725_bearing  << endl;
// cout << "AUV_ST725_range    = " << AUV_ST725_range    << endl;
// cout << "AUV_ST725_strength = " << AUV_ST725_strength << endl;

```

```

// initial code development prior to porting to dynamics.C

// Print hostname of PDU (revision in network.round version)
// hp = gethostbyaddr((char *) &inaddr, sizeof(struct in_addr), AF_INET);
// cout << "viewer: Host name: " << hp->h_name << endl;

// don't forget freePDU or get a memory leak!
// articulated parameters are also freed

freePDU (local_PDU);

// cout << "viewer: freePDU (local_PDU) called for this PDU" << endl;

} // end while infinite loop

// cout << "viewer: DIS net_read portion complete, now update scene graph."
// << endl;

current_clock = clock ();
delta_clock = current_clock - time_of_PDU_receipt;
delta_time = (double) delta_clock / CLOCKS_PER_SEC;

// cout << "viewer: current_clock = " << current_clock
// << ", time_stamp_of_current_PDU = " << time_stamp_of_current_PDU
// << ", time_of_PDU_receipt = " << time_of_PDU_receipt
// << ", PDU delta_time = " << delta_time << endl;

if ((delta_time >= 0.0) && (delta_time <= 5.0)) // update positions, pos-
tures
{
    if (DEADRECKON == TRUE)
    {
        delta_x = AUV_x_dot * delta_time;
        delta_y = AUV_y_dot * delta_time;
        delta_z = AUV_z_dot * delta_time;
        delta_phi = AUV_phi_dot * delta_time;
        delta_theta = AUV_theta_dot * delta_time;
        delta_psi = AUV_psi_dot * delta_time;
    }
    else
    {
        delta_x = 0.0;
        delta_y = 0.0;
        delta_z = 0.0;
        delta_phi = 0.0;
        delta_theta = 0.0;
        delta_psi = 0.0;
    }

    // cout << "viewer: DeadReckon_Callback: "
    // << " PDU delta_time = " << delta_time << endl;
    // cout << " AUV_phi = " << degrees (AUV_phi) << endl;
    // << ", AUV_phi_dot = " << degrees (AUV_phi_dot) << endl;
    // cout << " AUV_theta = " << degrees (AUV_theta) << endl;
    // << ", AUV_theta_dot = " << degrees (AUV_theta_dot) << endl;
    // cout << " delta_x = " << delta_x << endl;
    // cout << " delta_y = " << delta_y << endl;
    // cout << " delta_z = " << delta_z << endl;
    // cout << endl;

    // save current AUV position for next time the camera is repositioned
    AUV_x_prior = AUV_x;
    AUV_y_prior = AUV_y;
    AUV_z_prior = AUV_z;

    // graphics rendering problems: psi, rudders are opposite

```

```

// update overall AUV posture (both position & orientation)
AUV_position_node->translation.setValue ( AUV_x      + delta_x,
                                           - (AUV_y      + delta_y),
                                           - (AUV_z      + delta_z));
rotate_AUV_z->angle.setValue              ( - (AUV_psi   + delta_psi));
rotate_AUV_y->angle.setValue              ( - (AUV_theta + delta_theta));
rotate_AUV_x->angle.setValue              ( AUV_phi    + delta_phi);

// update AUV rudder & plane orientations

rotate_AUV_bow_rudders->angle.setValue ( radians(AUV_delta_rudder));
rotate_AUV_stern_rudders->angle.setValue ( - radians(AUV_delta_rudder));
ans(AUV_delta_rudder));
rotate_AUV_bow_planes->angle.setValue ( - radians(AUV_delta_planes));
rotate_AUV_stern_planes->angle.setValue ( radians(AUV_delta_planes));
ans(AUV_delta_planes));

// cout << "AUV_delta_rudder = " << AUV_delta_rudder
//      << ", radians (AUV_delta_rudder) = "
//      << radians (AUV_delta_rudder)
//      << endl;
// cout << "AUV_delta_planes= " << AUV_delta_planes
//      << ", radians (AUV_delta_planes) = "
//      << radians (AUV_delta_planes)
//      << endl;

if (fabs(AUV_stbd_rpm -
         (conePropellerWakeStbd->height.getValue() * 100.0/24.0)) > 10.0)
// ensure needed
{
    conePropellerWakeStbd->height = AUV_stbd_rpm /100.0*24.0;
    conePropellerWakeStbd->bottomRadius = fabs(AUV_stbd_rpm) /100.0* 6.0;
}
if (fabs(AUV_port_rpm -
         (conePropellerWakePort->height.getValue() * 100.0/24.0)) > 10.0)
// ensure needed
{
    conePropellerWakePort->height = AUV_port_rpm /100.0*24.0;
    conePropellerWakePort->bottomRadius = fabs (AUV_port_rpm)/100.0* 6.0;
}
// always recalculate thrusters
{
    thrusterWakeFV->height = - AUV_bow_vertical * 2.0;
    thrusterWakeFV->bottomRadius = AUV_bow_vertical / 4.0;
    if (AUV_bow_vertical < 0.0)
        // bottomsideBow, negative volts push up (negative direction)
        whichWakeFV->whichChild = 1;
        // topsideBow, positive volts push down (positive direction)
    else whichWakeFV->whichChild = 0;

    topsideBow-> translation.setValue(0, TOPHALF+AUV_bow_vertical,0);
    bottomsideBow->translation.setValue(0,BOTTOMHALF+AUV_bow_vertical,0);
}
// always recalculate thrusters
{
    thrusterWakeAV->height = - AUV_stern_vertical * 2.0;
    thrusterWakeAV->bottomRadius = AUV_stern_vertical / 4.0;
    if (AUV_stern_vertical < 0.0)
        // bottomsideStern, negative volts push up (negative direction)
        whichWakeAV->whichChild = 1;
        // topsideStern, positive volts push down(positive direction)
    else whichWakeAV->whichChild = 0;

    topsideStern->translation.setValue(0, TOPHALF+AUV_stern_vertical,0);
    bottomsideStern->translation.setValue(0,BOTTOMHALF+AUV_stern_vertical,0);
}
// always recalculate thrusters
{

```

```

thrusterWakeFH->height      = - AUV_bow_lateral * 2.0;
thrusterWakeFH->bottomRadius =  AUV_bow_lateral / 4.0;
if (AUV_bow_lateral < 0.0)
    // rightsideBow, negative volts push left (negative direction)
    whichWakeFH->whichChild = 1;
    // leftsideBow, positive volts push right (positive direction)
else whichWakeFH->whichChild = 0;

leftsideBow->translation.setValue (0, LEFT_HALF+AUV_bow_lateral, 0);
rightsideBow->translation.setValue (0, RIGHTHALF+AUV_bow_lateral, 0);
}
// always recalculate thrusters
{
    thrusterWakeAH->height      = - AUV_stern_lateral * 2.0;
    thrusterWakeAH->bottomRadius =  AUV_stern_lateral / 4.0;
    if (AUV_stern_lateral < 0.0)
        // rightsideStern, negative volts push left (negative direction)
        whichWakeAH->whichChild = 1;
        // leftsideStern, positive volts push right (positive direction)
    else whichWakeAH->whichChild = 0;

    leftsideStern->translation.setValue(0,LEFT_HALF+AUV_stern_lateral,0);
    rightsideStern->translation.setValue(0,RIGHTHALF+AUV_stern_lateral,0);
}
// Process ST725 Data for Cone Representation
if ((fabs(AUV_ST725_range - coneSonarST725->height.getValue()) > 0.0) ||
    (AUV_ST725_range < 0.0))
    // ensure needed
    {
        if (AUV_ST725_range > 0.0)
            {
                coneSonarST725->height      = fabs (AUV_ST725_range);
                coneSonarST725->bottomRadius = fabs (AUV_ST725_range) / 60.0;
                // 1 degree
                xfConeSonarST725->translation.setValue ( 0.0,
                                                         - (AUV_ST725_range / 2.0),
                                                         0.0);
                sonar_cone_color725->ambientColor.setValue ( 0.8, 0.0, 0.8 );
                sonar_cone_color725->diffuseColor.setValue ( 0.8, 0.0, 0.8 );
                sonar_cone_color725->specularColor.setValue ( 0.8, 0.0, 0.8 );
                ST725Complexity->value.setValue ( ST725Complexity_on );
            }
        else if (AUV_ST725_range < 0.0)
            {
                coneSonarST725->height      = 0.001;
                coneSonarST725->bottomRadius = 0.0001;
                xfConeSonarST725->translation.setValue ( 0.0, 0.0, 0.0);
                sonar_cone_color725->ambientColor.setValue ( 0.0, 0.0, 0.0 );
                sonar_cone_color725->diffuseColor.setValue ( 0.0, 0.0, 0.0 );
                sonar_cone_color725->specularColor.setValue ( 0.7, 0.7, 0.7 );
                ST725Complexity->value.setValue ( ST725Complexity_off );
            }
        else // zero range returned, draw full-size no-detection cone
            {
                coneSonarST725->height      = 32.808;
                coneSonarST725->bottomRadius = 0.5468; // bottom of cone max range
                xfConeSonarST725->translation.setValue ( 0.0, -16.404, 0.0);
                sonar_cone_color725->ambientColor.setValue ( 0.0, 0.0, 0.0 );
                sonar_cone_color725->diffuseColor.setValue ( 0.0, 0.0, 0.0 );
                sonar_cone_color725->specularColor.setValue ( 0.7, 0.7, 0.7 );
                ST725Complexity->value.setValue ( ST725Complexity_off );
            }
    }

    rotConeSonarST725->angle = radians ( -AUV_ST725_bearing );
}

// Process ST1000 Data for Cone Representation

```

```

if ((fabs(AUV_ST1000_range - coneSonarST1000->height.getValue())>0.0)||
    (AUV_ST1000_range < 0.0))
// ensure needed
{
    if (AUV_ST1000_range > 0.0)
    {
        coneSonarST1000->height = fabs (AUV_ST1000_range);
        coneSonarST1000->bottomRadius = fabs (AUV_ST1000_range) / 60.0;
        // 1 degree
        xfConeSonarST1000->translation.setValue ( 0.0,
            - (AUV_ST1000_range / 2.0),
            0.0);
        sonar_cone_color1000->ambientColor.setValue ( 1.0, 0.0, 0.15 );
        sonar_cone_color1000->diffuseColor.setValue ( 1.0, 0.0, 0.15 );
        sonar_cone_color1000->specularColor.setValue ( 1.0, 0.0, 0.15 );
        ST1000Complexity->value.setValue ( ST1000Complexity_on );
    }
    else if (AUV_ST1000_range < 0.0)
    {
        coneSonarST1000->height = 0.001;
        coneSonarST1000->bottomRadius = 0.0001;
        xfConeSonarST1000->translation.setValue ( 0.0, 0.0, 0.0);
        sonar_cone_color1000->ambientColor.setValue ( 0.8, 0.8, 0.0 );
        sonar_cone_color1000->diffuseColor.setValue ( 0.8, 0.8, 0.0 );
        sonar_cone_color1000->specularColor.setValue ( 0.8, 0.8, 0.0 );
        ST1000Complexity->value.setValue ( ST1000Complexity_off );
    }
    else // zero range returned, draw full-size no-detection cone
    {
        coneSonarST1000->height = 32.808;
        coneSonarST1000->bottomRadius = 0.5468;
        xfConeSonarST1000->translation.setValue ( 0.0, -16.404, 0.0);
        sonar_cone_color1000->ambientColor.setValue ( 0.8, 0.8, 0.0 );
        sonar_cone_color1000->diffuseColor.setValue ( 0.8, 0.8, 0.0 );
        sonar_cone_color1000->specularColor.setValue ( 0.8, 0.8, 0.0 );
        ST1000Complexity->value.setValue ( ST1000Complexity_off );
    }

    if (TRACE) {
        cout << "xfConeSonarST1000->translation.x = "
            << xfConeSonarST1000->translation.getValue()[0] << endl;
        cout << "xfConeSonarST1000->translation.y = "
            << xfConeSonarST1000->translation.getValue()[1] << endl;
        cout << "xfConeSonarST1000->translation.z = "
            << xfConeSonarST1000->translation.getValue()[2] << endl;
    }

    rotConeSonarST1000->angle = radians ( -AUV_ST1000_bearing );
}

} // end processing current AUV PDU

else if (PDU_overdue == FALSE) // reset vehicle position, update scene graph
{
    PDU_overdue = TRUE; // over 5 seconds elapsed since last PDU

    // restore latest valid AUV posture (both position & orientation)
    AUV_position_node->translation.setValue ( AUV_x,
        - AUV_y,
        - AUV_z);
    rotate_AUV_z->angle.setValue ( - AUV_psi);
    rotate_AUV_y->angle.setValue ( - AUV_theta);
    rotate_AUV_x->angle.setValue ( AUV_phi);

    // thrusters
    AUV_bow_vertical = 0.0;
    AUV_stern_vertical = 0.0;
    AUV_bow_lateral = 0.0;

```

```

AUV_stern_lateral = 0.0;

thrusterWakeFV->height = 0.0;
thrusterWakeFV->bottomRadius = 0.0;
thrusterWakeAV->height = 0.0;
thrusterWakeAV->bottomRadius = 0.0;
thrusterWakeFH->height = 0.0;
thrusterWakeFH->bottomRadius = 0.0;
thrusterWakeAH->height = 0.0;
thrusterWakeAH->bottomRadius = 0.0;

AUV_ST1000_bearing = 0;
AUV_ST1000_range = -0.001;
AUV_ST1000_strength = 0;
AUV_ST725_bearing = 0;
AUV_ST725_range = -0.001;
AUV_ST725_strength = 0;

coneSonarST725->height = fabs(AUV_ST725_range);
coneSonarST725->bottomRadius = fabs(AUV_ST725_range) / 60.0; //1 degree
coneSonarST1000->height = fabs(AUV_ST1000_range);
coneSonarST1000->bottomRadius = fabs(AUV_ST1000_range) / 60.0; //1 degree
xfConeSonarST725->translation.setValue ( 0.0, 0.0, 0.0);
xfConeSonarST1000->translation.setValue ( 0.0, 0.0, 0.0);

if (TRACE) {
    cout << "xfConeSonarST1000->translation.x = "
          << xfConeSonarST1000->translation.getValue()[0] << endl;
    cout << "xfConeSonarST1000->translation.y = "
          << xfConeSonarST1000->translation.getValue()[1] << endl;
    cout << "xfConeSonarST1000->translation.z = "
          << xfConeSonarST1000->translation.getValue()[2] << endl;
}

AUV_port_rpm = 0.0;
AUV_stbd_rpm = 0.0;

conePropellerWakePort->height = 0.0;
conePropellerWakePort->bottomRadius = 0.0;
conePropellerWakeStbd->height = 0.0;
conePropellerWakeStbd->bottomRadius = 0.0;

if (DEADRECKON == TRUE)
{
    cout << "viewer: DeadReckon_Callback: "
          << "PDU delta_time = " << delta_time << ", "
          << endl;
    cout << "viewer position/posture reset to last received PDU." << endl;
    cout << endl;
    cout << flush;
}

}

if ((delta_time >= 0.0) && (delta_time <= 5.0)) // force camera update
{
    // camera control - - - - -

    priorAUVPosition = SbVec3f ( AUV_x_prior, -AUV_z_prior, AUV_y_prior );
    currentAUVPosition = SbVec3f ( AUV_x * METERS_PER_FT,
                                   -AUV_z * METERS_PER_FT,
                                   AUV_y * METERS_PER_FT );
    aheadOfAUVPosition = SbVec3f ( 5.0 * cos ( AUV_psi ),
                                    2.0 * sin ( AUV_theta ),
                                    5.0 * sin ( AUV_psi ) + 0.5);
    behindAUVPosition = SbVec3f ( -5.0 * cos ( AUV_psi ),
                                   -2.0 * sin ( AUV_theta ),
                                   -3.0 * sin ( AUV_theta ),
                                   -5.0 * sin ( AUV_psi ) + 0.5);
}

```



```

switch ( whichCamera ) // reposition appropriate camera as needed
{
case CAMERA_FREE: // FREE means no modifications to camera position
    priorCameraPosition = PerspectiveCameraFree->position.getValue ();
    priorCameraOffset = priorCameraPosition - priorAUVPosition;
    currentCameraPosition = priorCameraPosition;
    PerspectiveCameraFree->orientation.getValue(orientationRotationAxis,
                                                orientationRotationAngle);
    break;

case CAMERA_TO_AUV: // retain camera pos'n relative to new AUV position
    priorCameraPosition = PerspectiveCameraToAUV->position.getValue ();
    priorCameraOffset = priorCameraPosition - priorAUVPosition;
// verify here
    priorCameraPosition = PerspectiveCameraToAUV->position.getValue ();
    // PerspectiveCameraToAUV->position.setValue ( currentAUVPosition
    // + standardCameraOffset );
    PerspectiveCameraToAUV->pointAt ( currentAUVPosition );
    currentCameraPosition = PerspectiveCameraToAUV->position.getValue ();
    PerspectiveCameraToAUV->orientation.getValue (orientationRotationAxis,
                                                orientationRotationAngle );
    break;

case CAMERA_FROM_AUV: // retain camera position looking from AUV pos.
    priorCameraPosition = PerspectiveCameraFromAUV->position.getValue ();
    priorCameraOffset = priorCameraPosition - priorAUVPosition;
// verify here
    currentCameraPosition = ( currentAUVPosition );
    PerspectiveCameraFromAUV->position.setValue ( currentAUVPosition
                                                +AUVCameraOffset
                                                +behindAUVPosition );
    PerspectiveCameraFromAUV->pointAt ( currentAUVPosition
                                       +AUVCameraOffset
                                       +aheadOfAUVPosition );
    PerspectiveCameraFromAUV->orientation.getValue (orientationRotationAxis,
                                                orientationRotationAngle );
    break;

} // end switch ( whichCamera )

/*
// print out all camera parameters
cout << endl;
cout << " AUV_position = <" << (AUV_x)
    << ", " << (AUV_y)
    << ", " << (AUV_z) << "> "
    << endl;

cout << "currentAUVPosition = <" << currentAUVPosition [0]
    << ", " << currentAUVPosition [1]
    << ", " << currentAUVPosition [2]
    << "> meters, shifted to light coordinate system"
    << endl;

cout << "aheadOfAUVPosition = <" << aheadOfAUVPosition [0]
    << ", " << aheadOfAUVPosition [1]
    << ", " << aheadOfAUVPosition [2]
    << "> meters" // shifted to light coordinate system
    << endl;

cout << "delta_AUV_position = <" << (AUV_x - AUV_x_prior)
    << ", " << (AUV_y - AUV_y_prior)
    << ", " << (AUV_z - AUV_z_prior) << "> "
    << endl;

cout << "delta_CameraPosition ="
    << " <" << (currentCameraPosition [0] - priorCameraPosition [0])
    << ", " << (currentCameraPosition [1] - priorCameraPosition [1])
    << ", " << (currentCameraPosition [2] - priorCameraPosition [2])
    << "> " << endl;

```

```

        cout << "priorCameraPosition ="
        << " <" << priorCameraPosition [0]
        << ", " << priorCameraPosition [1]
        << ", " << priorCameraPosition [2] << ">" << endl;
    cout << "priorCameraOffset ="
        << " <" << priorCameraOffset [0]
        << ", " << priorCameraOffset [1]
        << ", " << priorCameraOffset [2] << ">" << endl;
    cout << "currentCameraPosition ="
        << " <" << currentCameraPosition [0]
        << ", " << currentCameraPosition [1]
        << ", " << currentCameraPosition [2] << ">" << endl;
    cout << "orientationRotation ="
        << " <" << orientationRotationAxis [0]
        << ", " << orientationRotationAxis [1]
        << ", " << orientationRotationAxis [2]
        << ", " << degrees (orientationRotationAngle) << ">" << endl;
*/
    // camera control complete - - - - -
}

// cout << "viewer: end of DIS_Redraw_Callback ()" << endl;

return;
}
//-----//

// called on an exit condition via a call to atexit (DIS_net_close) in main
static void DIS_net_close ()
{
    cout << "viewer: DIS_net_close ();" << endl;
    net_close ();
    DIS_port_open = FALSE;
}
//-----//

////////////////////////////////////

//
// This is called by the Color Editor whenever the color
// has changed. The userData is set by main() in the call
// to SoXtColorEditor::addColorChangedCallback.
//
void
colorEditorCB( void *userData, const SbColor *rgbCallbackData )
{
    SoXtRenderArea *renderArea = (SoXtRenderArea *) userData;
    renderArea->setBackgroundColor( *rgbCallbackData );
}

////////////////////////////////////

SoSeparator * readFile(const char *filename) // Inventor Mentor p. 284
{
    // Open the input file
    SoInput mySceneInput;
    if (!mySceneInput.openFile(filename))
    {
        cout << "Cannot open file " << filename << endl;
        return NULL;
    }
    // Read the whole file into the database
    SoSeparator * myGraph = SoDB::readAll(&mySceneInput);
    if (myGraph == NULL)
    {
        cout << "Problem reading file " << filename << endl;
    }
}

```

```

        return NULL;
    }
    mySceneInput.closeFile ();
    return myGraph;
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void initialize_globals ()
{
    // multicast port & group
    bzero (port, 6);
    strncpy (port, "3111", 4);
    bzero (group, 30);
    strncpy (group, "224.2.244.141", 13); // GROUP

    // strncpy (group, "224.2.237.170", 13); // GROUP
    // 3111 is npsnet 'standard' port
    // #define DEFAULT_PORT = "3111";
    // #define DEFAULT_GROUP = "224.2.121.93";

    // initialize materials
    silver->ambientColor.setValue ( .2, .2, .2 );
    silver->diffuseColor.setValue ( .6, .6, .6 );
    silver->specularColor.setValue ( .5, .5, .5 );
    silver->shininess = .5;

    gold->ambientColor.setValue ( .4, .2, .0 );
    gold->diffuseColor.setValue ( .9, .5, .0 );
    gold->specularColor.setValue ( .7, .7, .0 );
    gold->shininess = .6;

    brass->ambientColor.setValue ( 0.329412, 0.223529, 0.027451 );
    brass->diffuseColor.setValue ( 0.780392, 0.568627, 0.113725 );
    brass->specularColor.setValue ( 0.992157, 0.941176, 0.807843 );
    brass->shininess = 0.21794872;

    chrome->ambientColor.setValue ( 0.25, 0.25, 0.25 );
    chrome->diffuseColor.setValue ( 0.4, 0.4, 0.4 );
    chrome->specularColor.setValue ( 0.774597, 0.774597, 0.774597 );
    chrome->shininess = 0.6;

    npsblue->ambientColor.setValue ( 0.0, 0.0, 1.0 );
    npsblue->diffuseColor.setValue ( 0.0, 0.0, 0.8 );
    npsblue->specularColor.setValue ( 0.0, 0.2, 1.0 );
    npsblue->shininess = 0.8;

    seagreen->ambientColor.setValue ( 0.0, 0.5, 0.0 );
    seagreen->diffuseColor.setValue ( 0.0, 0.5, 0.0 );
    seagreen->specularColor.setValue ( 0.0, 0.5, 0.0 );
    seagreen->shininess = 0.0;

    seasurface->ambientColor.setValue ( 0.3, 0.7, 0.5 );
    seasurface->diffuseColor.setValue ( 0.3, 0.7, 0.5 );
    seasurface->specularColor.setValue ( 0.3, 0.7, 0.5 );
    seasurface->shininess = 0.0;

    seacolor->ambientColor.setValue ( 0.1, 0.1, 0.5 );
    seacolor->diffuseColor.setValue ( 0.1, 0.1, 0.5 );
    seacolor->specularColor.setValue ( 0.1, 0.1, 0.5 );
    seacolor->shininess = 0.0;

    darkgreen->ambientColor.setValue ( 0.15, 0.20, 0.15 );
    darkgreen->diffuseColor.setValue ( 0.15, 0.20, 0.15 );
    darkgreen->specularColor.setValue ( 0.15, 0.20, 0.15 );
    darkgreen->shininess = 0.0;

    sonar_cone_color725->ambientColor.setValue ( 0.8, 0.0, 0.8 );

```

```

sonar_cone_color725->diffuseColor.setValue ( 0.8, 0.0, 0.8 );
sonar_cone_color725->specularColor.setValue ( 0.8, 0.0, 0.8 );
sonar_cone_color725->shininess = 0.0;

sonar_cone_color1000->ambientColor.setValue ( 1.0, 0.0, 0.15 );
sonar_cone_color1000->diffuseColor.setValue ( 1.0, 0.0, 0.15 );
sonar_cone_color1000->specularColor.setValue ( 1.0, 0.0, 0.15 );
sonar_cone_color1000->shininess = 0.0;

wires->style = SoDrawStyle::LINES;

// Units of distance initialized
MetersToFeet->scaleFactor.setValue ( METERS_PER_FT, METERS_PER_FT,
                                     METERS_PER_FT );
InchesToFeet->scaleFactor.setValue ( 12.0/1.0, 12.0/1.0, 12.0/1.0 );
FeetToInches->scaleFactor.setValue ( 1.0/12.0, 1.0/12.0, 1.0/12.0 );

MineFieldTransform->translation.setValue( 3280.8/2.0, -3280.8/2.0, 0.0);
MineFieldScale->scaleFactor.setValue ( 3280.8/400.0, 3280.8/400.0, 1.0);

// initial position of AUV
AUV_x      = 5.0;
AUV_y      = 5.0;
AUV_z      = 2.0;

AUV_ST1000_bearing = 0;
AUV_ST1000_range   = -0.001;
AUV_ST1000_strength = 0;
AUV_ST725_bearing  = 0;
AUV_ST725_range    = -0.001;
AUV_ST725_strength = 0;

wakeComplexity->value = 0.2;
ST1000Complexity->value = ST1000Complexity_off;
ST725Complexity->value = ST725Complexity_off;

return;
} // end initialize_globals ()

////////////////////////////////////
////////////////////////////////////

void parse_command_line_flags (int argc, char ** argv)
    // command line arguments
{
    int i;

    // cout << "[parse_command_line_flags start: # arguments = " << argc << "]"
    // << endl;
    // cout << "{ ";
    // for (i = 0; i < argc; i++) cout << argv [i] << " ";
    // cout << "]" << endl;

    for (i = 1; i < argc; i++)
    {
        if ((strcmp (argv[i-1], "CIRCLE-FILE") != 0) &&
            (strcmp (argv[i-1], "CIRCLEFILE") != 0) &&
            (strcmp (argv[i-1], "CIRCLE") != 0) &&
            (strcmp (argv[i-1], "CYLINDER-FILE") != 0) &&
            (strcmp (argv[i-1], "CYLINDERFILE") != 0) &&
            (strcmp (argv[i-1], "CYLINDER") != 0) &&
            (strcmp (argv[i-1], "WORLD-FILE") != 0) &&
            (strcmp (argv[i-1], "WORLDFILE") != 0) &&
            (strcmp (argv[i-1], "WORLD") != 0))
            for (int index = 0; index <= strlen (argv[i]); index++) // uppercase
                argv[i] [index] = toupper (argv[i] [index]);

        if ((strcmp (argv[i], "PORT") == 0) ||

```

```

        (strcmp (argv[i], "-PORT") == 0) ||
        (strcmp (argv[i], "P") == 0) ||
        (strcmp (argv[i], "-P") == 0))
    {
        if ( i+1 >= argc )
            cout << "Insufficient parameters for PORT" << endl;
        else
        {
            cout << "[" << argv[i] << " " << argv[i+1] << "]" << endl;
            strcpy (port, argv[i+1]);
            i++;
        }
    }
    else if ((strcmp (argv[i], "GROUP") == 0) ||
             (strcmp (argv[i], "-GROUP") == 0) ||
             (strcmp (argv[i], "G") == 0) ||
             (strcmp (argv[i], "-G") == 0) ||
             (strcmp (argv[i], "ADDRESS") == 0) ||
             (strcmp (argv[i], "-ADDRESS") == 0) ||
             (strcmp (argv[i], "A") == 0) ||
             (strcmp (argv[i], "-A") == 0))
    {
        if ( i+1 >= argc )
            cout << "Insufficient parameters for GROUP ADDRESS" << endl;
        else
        {
            cout << "[" << argv[i] << " " << argv[i+1] << "]" << endl;
            strcpy (group, argv[i+1]);
            i++;
            cout << "[" << argv[i] << " " << argv[i+1] << "]" << endl;
        }
    }
    else if ((strcmp (argv[i], "PRINTDIALOG") == 0) ||
             (strcmp (argv[i], "-PRINTDIALOG") == 0) ||
             (strcmp (argv[i], "PRINT") == 0) ||
             (strcmp (argv[i], "-PRINT") == 0))
    {
        PRINTDIALOG = TRUE;
        cout << "[" << argv[i] << "]" << endl;
    }
    else if ((strcmp (argv[i], "NOPRINTDIALOG") == 0) ||
             (strcmp (argv[i], "-NOPRINTDIALOG") == 0) ||
             (strcmp (argv[i], "NOPRINT") == 0) ||
             (strcmp (argv[i], "-NOPRINT") == 0))
    {
        PRINTDIALOG = FALSE;
        cout << "[" << argv[i] << "]" << endl;
    }
    else if ((strcmp (argv[i], "TEXTURE") == 0) ||
             (strcmp (argv[i], "-TEXTURE") == 0) ||
             (strcmp (argv[i], "TEXTURE-ON") == 0) ||
             (strcmp (argv[i], "-TEXTURE-ON") == 0) ||
             (strcmp (argv[i], "T") == 0) ||
             (strcmp (argv[i], "-T") == 0))
    {
        TEXTURE = TRUE;
        cout << "[" << argv[i] << "]" << endl;
    }
    else if ((strcmp (argv[i], "MINEFIELD") == 0) ||
             (strcmp (argv[i], "-MINE-FIELD") == 0))
    {
        MINEFIELD = TRUE; // special color scheme
        cout << "[" << argv[i] << "]" << endl;
    }
    else if ((strcmp (argv[i], "NOTEXTURE") == 0) ||
             (strcmp (argv[i], "-NOTEXTURE") == 0) ||
             (strcmp (argv[i], "TEXTURE-OFF") == 0) ||
             (strcmp (argv[i], "-TEXTURE-OFF") == 0))
    {

```

```

        (strcmp (argv[i], "NO-TEXTURE") == 0) ||
        (strcmp (argv[i], "-NO-TEXTURE") == 0))
    {
        TEXTURE = FALSE;
        cout << "[" << argv[i] << "]" << endl;
    }
    else if ((strcmp (argv[i], "SCREENDOOR") == 0) ||
             (strcmp (argv[i], "-SCREENDOOR") == 0) ||
             (strcmp (argv[i], "SCREEN-DOOR") == 0) ||
             (strcmp (argv[i], "-SCREEN-DOOR") == 0))
    {
        SCREENDOOR = TRUE;
        cout << "[" << argv[i] << "]" << endl;
    }
    else if ((strcmp (argv[i], "LIGHTSOUT") == 0) ||
             (strcmp (argv[i], "-LIGHTSOUT") == 0) ||
             (strcmp (argv[i], "LIGHTS-OUT") == 0) ||
             (strcmp (argv[i], "-LIGHTS-OUT") == 0) ||
             (strcmp (argv[i], "NOLIGHTS") == 0) ||
             (strcmp (argv[i], "-NOLIGHTS") == 0) ||
             (strcmp (argv[i], "NO-LIGHTS") == 0) ||
             (strcmp (argv[i], "-NO-LIGHTS") == 0))
    {
        LIGHTSOUT = TRUE;
        cout << "[" << argv[i] << "]" << endl;
    }
    else if ((strcmp (argv[i], "FILE") == 0) ||
             (strcmp (argv[i], "-FILE") == 0) ||
             (strcmp (argv[i], "F") == 0) ||
             (strcmp (argv[i], "-F") == 0))
    {
        if ( i+1 >= argc )
            cout << "Insufficient parameters for FILE filename.iv"
                << endl;
        else
        {
            cout << "[" << argv[i] << " " << argv[i+1] << "]" << endl;
            command_line_node = new SoSeparator;
            command_line_node = readFile (argv[i+1]);
            root-> addChild ( command_line_node );
            i++;
        }
    }
    else if ((strcmp (argv[i], "BACKGROUNDCOLORDIALOG") == 0) ||
             (strcmp (argv[i], "-BACKGROUNDCOLORDIALOG") == 0))
    {
        BACKGROUNDCOLORDIALOG = TRUE;
        cout << "[" << argv[i] << "]" << endl;
    }
    else if ((strcmp (argv[i], "NOBACKGROUNDCOLORDIALOG") == 0) ||
             (strcmp (argv[i], "-NOBACKGROUNDCOLORDIALOG") == 0))
    {
        BACKGROUNDCOLORDIALOG = FALSE;
        cout << "[" << argv[i] << "]" << endl;
    }
    else if ((strcmp (argv[i], "DEADRECKON") == 0) ||
             (strcmp (argv[i], "-DEADRECKON") == 0) ||
             (strcmp (argv[i], "DR") == 0) ||
             (strcmp (argv[i], "-DR") == 0))
    {
        DEADRECKON = TRUE;
        cout << "[" << argv[i] << "]" << endl;
    }
    else if ((strcmp (argv[i], "NODEADRECKON") == 0) ||
             (strcmp (argv[i], "-NODEADRECKON") == 0) ||
             (strcmp (argv[i], "NO-DEADRECKON") == 0) ||
             (strcmp (argv[i], "-NO-DEADRECKON") == 0) ||
             (strcmp (argv[i], "NODR") == 0) ||
             (strcmp (argv[i], "-NODR") == 0))
    {

```



```

        (strcmp (argv[i], "SURFACE") == 0) ||
        (strcmp (argv[i], "-SURFACE") == 0))
    {
        TESTTANKSURFACE = TRUE;
        cout << "[" << argv[i] << "]" << endl;
    }
else if ((strcmp (argv[i], "CIRCLE-FILE") == 0) ||
        (strcmp (argv[i], "CIRCLEFILE") == 0) ||
        (strcmp (argv[i], "CIRCLE") == 0) ||
        (strcmp (argv[i], "CYLINDERFILE") == 0) ||
        (strcmp (argv[i], "CYLINDER-FILE") == 0) ||
        (strcmp (argv[i], "CYLINDER") == 0) )
    {
        if (READCIRCLEFILE)
        {
            cout << "Error - cannot open more than one circle file: "
                << argv[i+1] << endl;
            exit (-1);
        }
        if (i + 1 >= argc)
            cout << "Insufficient parameters for CIRCLE-FILE" << endl;
        else
        {
            i++;
            char newfilename [50];
            strcpy (newfilename, argv[i]);
            circleFile.open(newfilename,ios::in);
            if (!circleFile)
            {
                strcpy (newfilename, "../tactical/");
                strcat (newfilename, argv[i]);
                circleFile.open(newfilename,ios::in);
            }
            if (!circleFile)
            {
                cout << "Error opening circle file: " << argv[i] << endl;
                exit (-1);
            }
            else
            {
                cout << "[Circle File " << newfilename << " opened]"
                    << endl;
                READCIRCLEFILE = TRUE;
            }
        }
    }
else if ((strcmp (argv[i], "WORLD-FILE") == 0) ||
        (strcmp (argv[i], "WORLDFILE") == 0) ||
        (strcmp (argv[i], "WORLD") == 0))
    {
        if (i + 1 >= argc)
            cout << "Insufficient parameters for WORLD-FILE" << endl;
        else
        {
            i++;
            SoSeparator * worldSeparator = readFile (argv[i]);
            if (!worldSeparator)
            {
                char newfilename [50];
                strcpy (newfilename, "../viewer/");
                strcat (newfilename, argv[i]);
                worldSeparator = readFile (newfilename);
            }
            if (worldSeparator)
            {
                scenery->addChild (worldSeparator);
                cout << "[World File " << argv[i] << " opened]" << endl;
            }
        }
    }

```



```

        else
        {
            cout << "Unable to locate world-file " << argv[i] << endl;
            exit (-1);
        }
    }
    else cout << "Unrecognized command line parameter: " << argv[i]
           << ", ignored." << endl;

} // end for loop through command line parameters

// cout << "[parse_command_line_flags complete]" << endl;

return;

} // end parse_command_line_flags ()

/////////////////////////////////////////////////////////////////
// Keypress event callbacks: Inventor Mentor p. 464-465
/*
void key_C_PressCB (void *, SoAction *)
{
    // cout << "key_C_PressCB entered, key_event->getKey () = "
    //                                     << key_event->getKey () << endl;

    // Check for C key being pressed:
    if (key_event->getKey () == SoKeyboardEvent::C)
    {
        whichCamera = (whichCamera++) % 3;
        cout << "key_C_PressCB camera toggled to " << whichCamera << endl;
    }
} // key_C_PressCB complete

// Keypress event callbacks: Inventor Mentor p. 266-267, SceneViewer.c++
SbBool key_C_PressCB (void * userData, XAnyEvent * event)
{
    SbBool handled = FALSE;
    // SoXtRenderArea * myRenderArea = (SoXtRenderArea *) userData;

    cout << "key_C_PressCB entered" << endl;

    // Check for C key being pressed:
    if (event->type == KeyPress)
    {
        if (XLookupKeysym ((XKeyEvent *) event, 0) == XK_C)
        {
            whichCamera = (whichCamera++) % 3;
            cout << "key_C_PressCB camera toggled to " << whichCamera << endl;
            handled = TRUE;
        }
    }
    return handled;
} // key_C_PressCB complete
*/

/////////////////////////////////////////////////////////////////

void main ( int argc, char ** argv )
{
    // Initialize Inventor and Xt - these steps MUST be first calls
    // in main, without exception, or a mystery crash results.
    Widget ViewerWindowWidget = SoXt::init(argv[0]);
    if ( ViewerWindowWidget == NULL )

```

```

{
    cout << "viewer: ViewerWindowWidget == NULL on startup, exiting."
          << endl;
    exit( 1 );
}
cout << "viewer: ViewerWindowWidget added" << endl;

initialize_globals ();

parse_command_line_flags (argc, argv);

// port and group can change by command line switches
cout << "multicast port = " << port
      << ", address group = " << group << endl;

cout << "creating the scene graph: " ;

root = new SoSeparator;
root->ref();
cout << "root added" << endl;

// Keypress event callbacks: Inventor Mentor p. 265-267, class notes ch. 10
// SoEventCallback *key_C_EventCB = new SoEventCallback;
// key_C_EventCB->addEventCallback (SoKeyboardEvent::getClassTypeId(),
//                                  key_C_PressCB, root);
// root->addChild (key_C_EventCB);

// Keypress event callbacks: Inventor Mentor p. 465
// SoCallback *key_C_EventCB = new SoCallback ();
// key_C_EventCB->setCallback (key_C_PressCB);
// root->addChild (key_C_EventCB);

// correct for different coordinate system - not yet needed
// SoRotationXYZ * coordinateSystemFlip = new SoRotationXYZ;
// coordinateSystemFlip->angle.setValue ( M_PI );
// coordinateSystemFlip->axis.setValue ( SoRotationXYZ::X );
// root->addChild( coordinateSystemFlip );

SoRotationXYZ *ro0x = new SoRotationXYZ;
ro0x->angle.setValue ( 0.0 );
ro0x->axis.setValue (SoRotationXYZ::X);

SoRotationXYZ *ro90x = new SoRotationXYZ;
ro90x->angle.setValue (M_PI / 2.0);
ro90x->axis.setValue (SoRotationXYZ::X);

SoRotationXYZ *ro270x = new SoRotationXYZ;
ro270x->angle.setValue (3.0 * M_PI / 2.0);
ro270x->axis.setValue (SoRotationXYZ::X);

PerspectiveCameraFree      = new SoPerspectiveCamera;
PerspectiveCameraToAUV     = new SoPerspectiveCamera;
PerspectiveCameraFromAUV   = new SoPerspectiveCamera;

// can't put a group or separator above camera
// cameras using pointAt are 90 degrees twisted askew
// Create a camera SoSwitch node
SoSwitch * whichCameraSwitch = new SoSwitch;
root->addChild ( whichCameraSwitch );
whichCameraSwitch->addChild ( PerspectiveCameraFree      );
whichCameraSwitch->addChild ( PerspectiveCameraToAUV     );
whichCameraSwitch->addChild ( PerspectiveCameraFromAUV   );

// select camera
whichCameraSwitch->whichChild = whichCamera;

// Create a camera rotation correction SoSwitch node
SoSwitch * whichCameraCorrectionSwitch = new SoSwitch;

```

```

root->addChild ( whichCameraCorrectionSwitch );
whichCameraCorrectionSwitch->addChild ( ro0x );
whichCameraCorrectionSwitch->addChild ( ro270x );
whichCameraCorrectionSwitch->addChild ( ro270x );
whichCameraCorrectionSwitch->whichChild = whichCamera;

SoXtRenderArea * myRenderArea = new SoXtRenderArea (ViewerWindowWidget);

// Keypress event callbacks: Inventor Mentor p. 266-267
// myRenderArea->setEventCallback (key_C_PressCB, myRenderArea);

myRegion = new SbViewportRegion (myRenderArea->getSize ());

if (SCREENDOOR == TRUE)
    myRenderArea->setTransparencyType (SoGLRenderAction::SCREEN_DOOR);
else
    myRenderArea->setTransparencyType (SoGLRenderAction::SORTED_OBJECT_BLEND);

if (LIGHTSOUT == FALSE)
    root->addChild( new SoPointLight );

// 2.0 ivview/ivquicken fails on SoUnits nodes so SoScale nodes used instead
unitsfeet = new SoUnits;
unitsfeet->units.setValue ( SoUnits::FEET );
root->addChild( unitsfeet );
// root->addChild( MetersToFeet );

currentAUVPosition = SbVec3f ( AUV_x * METERS_PER_FT,
                              -AUV_z * METERS_PER_FT,
                              AUV_y * METERS_PER_FT );
aheadOfAUVPosition = SbVec3f ( 5.0 * sin ( AUV_psi ),
                              2.0 * sin ( AUV_theta ),
                              5.0 * cos ( AUV_psi ) + 0.5);
behindAUVPosition = SbVec3f ( -5.0 * sin ( AUV_psi ),
                              -2.0 * sin ( AUV_theta ),
                              -3.0 * sin ( AUV_theta ),
                              -5.0 * cos ( AUV_psi ) + 0.5);

// Free (unmodified) Camera
PerspectiveCameraFree->viewAll (root, *myRegion, 1.0); // global
PerspectiveCameraFree->aspectRatio.setValue (SO_ASPECT_VIDEO);

// Camera that keeps AUV in center
PerspectiveCameraToAUV->viewAll (root, *myRegion, 1.0); // global
PerspectiveCameraToAUV->aspectRatio.setValue (SO_ASPECT_VIDEO);
PerspectiveCameraToAUV->position.setValue
    ( currentAUVPosition + standardCameraOffset );
// PerspectiveCameraToAUV->orientation.setValue
//    ( SbRotation (1.0, 0.0, 0.0, M_PI / 2.0 ) );
PerspectiveCameraToAUV->pointAt ( currentAUVPosition );

// Camera that looks out from AUV in center
PerspectiveCameraFromAUV->viewAll (root, *myRegion, 1.0); // global
PerspectiveCameraFromAUV->aspectRatio.setValue (SO_ASPECT_VIDEO);
PerspectiveCameraFromAUV->position.setValue ( currentAUVPosition
                                             +AUVCameraOffset
                                             +behindAUVPosition );
// PerspectiveCameraFromAUV->orientation.setValue
//    (SbRotation (1.0, 0.0, 0.0, M_PI / 2.0 ));
PerspectiveCameraFromAUV->pointAt ( currentAUVPosition
                                   +AUVCameraOffset
                                   +aheadOfAUVPosition );

SoPickStyle * unpickablestylenode;
SoPickStyle * pickablestylenode;
unpickablestylenode = new SoPickStyle;
pickablestylenode = new SoPickStyle;

```

```

unpickablestylenode->style.setValue ( SoPickStyle::UNPICKABLE );
pickablestylenode->style.setValue ( SoPickStyle::SHAPE );

// create the terrain box
SoCube * backgroundCube = new SoCube;
backgroundCube->width = 400.0;
backgroundCube->height = 400.0;
backgroundCube->depth = 1.0;
SoTransform * xfbackgroundCube = new SoTransform;
xfbackgroundCube->translation.setValue(0.0, 0.0, -50.0);
SoTexture2 * overhangTexture = new SoTexture2;
overhangTexture->filename.setValue ("overhang.rgb");
SoSeparator * sepbackgroundCube = new SoSeparator;
sepbackgroundCube->addChild( pickablestylenode );
sepbackgroundCube->addChild( unitsfeet );
if (TEXTURE == TRUE)
    sepbackgroundCube->addChild( overhangTexture );
sepbackgroundCube->addChild( xfbackgroundCube );
if (MINEFIELD)
{
    sepbackgroundCube->addChild( seacolor );
    sepbackgroundCube->addChild( MineFieldTransform );
    sepbackgroundCube->addChild( MineFieldScale );
}
else sepbackgroundCube->addChild( darkgreen );

sepbackgroundCube->addChild( backgroundCube ); // remove cube here
root->addChild ( sepbackgroundCube );

root->addChild ( scenery ); // pickable objects go in this separator

// Read from circleFile and draw cylinders (if required)
if (READCIRCLEFILE)
    readCircles ();

// Jason with engine animation
SoSeparator * sepJason = new SoSeparator;
SoTransform * xfJason = new SoTransform;
SoSeparator * Jason = readFile ("Jason.iv");
xfJason->translation.setValue( -60.0, 40.0, - 25.0); // center of pattern
sepJason->addChild ( xfJason );
// animation from Inventor Mentor 13.6.Calculator.c++
// Set up the Jason transformations
SoRotationXYZ * danceRotate = new SoRotationXYZ;
danceRotate->angle.setValue ( 0.0 );
danceRotate->axis.setValue ( SoRotationXYZ::Z );
sepJason->addChild ( danceRotate );
SoTranslation * danceTranslate = new SoTranslation;
sepJason->addChild ( danceTranslate );
sepJason->addChild ( Jason );
root->addChild ( sepJason );
// Set up an engine to calculate the motion path:
// Theta is incremented using a time counter engine,
// and converted to radians using an expression in
// the calculator engine.
SoCalculator * calcXY = new SoCalculator;
SoTimeCounter * thetaCounter = new SoTimeCounter;
thetaCounter->max = 360;
thetaCounter->step = 1;
thetaCounter->frequency = 1 / 180.0; // 180 seconds for a full cycle
calcXY->a.connectFrom(&thetaCounter->output);
calcXY->expression.set1Value(0, "ta=a*M_PI/180"); // theta (radians)
calcXY->expression.set1Value(1, "tb=15*cos(ta)"); // r, z
calcXY->expression.set1Value(2, "td=tb*cos(ta)"); // x
calcXY->expression.set1Value(3, "te=tb*sin(ta)"); // y
calcXY->expression.set1Value(4, "oA=vec3f(td,te,tb)"); // vector output A
danceTranslate->translation.connectFrom(&calcXY->oA);
calcXY->expression.set1Value(5, "ob=2*ta"); // scalar output b

```

```

danceRotate->angle.connectFrom(&calcXY->ob);

// Oil Platform
SoSeparator * sepPlatform = new SoSeparator;
SoTransform * xfPlatform = new SoTransform;
SoSeparator * Platform = readFile ("Platform.iv");
xfPlatform->translation.setValue(-80.0, 50.0, -50.0);
sepPlatform->addChild ( xfPlatform );
sepPlatform->addChild( Platform );
scenery->addChild ( sepPlatform );

// Testtank
SoSeparator * sepTesttank = new SoSeparator;
SoTransform * xfTesttank = new SoTransform;
SoSeparator * Testtank = readFile ("testtank.iv");
if (TESTTANKSURFACE)
    xfTesttank->translation.setValue( 0.0, 0.0, -6.0);
else
    xfTesttank->translation.setValue( 0.0, 0.0, -50.0);
sepTesttank->addChild ( xfTesttank );
sepTesttank->addChild( Testtank );
scenery->addChild ( sepTesttank );

// Torpedo Tube
SoSeparator * sepTorpedoTube = new SoSeparator;
SoTransform * xfTorpedoTube = new SoTransform;
xfTorpedoTube->translation.setValue( 30.0, 20.0, -48.00 );
SoCylinder * TorpedoTube = new SoCylinder;
TorpedoTube->radius = 21.0 / 12.0; // 21" diameter
TorpedoTube->height = 10.0; // 10' length
TorpedoTube->parts = SoCylinder::SIDES; // let's drive through!
sepTorpedoTube->addChild ( xfTorpedoTube );
sepTorpedoTube->addChild( TorpedoTube );
scenery->addChild ( brass );
scenery->addChild ( sepTorpedoTube );

// Initialize for makeAUV
whichWakeFV->addChild (topsideBow);
whichWakeFV->addChild (bottomsideBow);
whichWakeFV->whichChild = 0; // default topsideBow
whichWakeAV->addChild (topsideStern);
whichWakeAV->addChild (bottomsideStern);
whichWakeAV->whichChild = 0; // default topsideStern
whichWakeFH->addChild (leftsideBow);
whichWakeFH->addChild (rightsideBow);
whichWakeFH->whichChild = 0; // default leftsideBow
whichWakeAH->addChild (leftsideStern);
whichWakeAH->addChild (rightsideStern);
whichWakeAH->whichChild = 0; // default leftsideStern

// Now get the AUV using makeAUV ();
SoSeparator * AUV_node = makeAUV ();
scenery->addChild ( AUV_node ); // AUV object creation routine above

// WriteAction writes scene graph to file
FILE * auv_iv_fp = fopen ("auv.iv", "w");
SoWriteAction writeaction;
writeaction.getOutput()-> setFilePointer (auv_iv_fp);
writeaction.apply (AUV_node);
fclose (auv_iv_fp);
cout << "writeaction.apply (AUV_node) => auv.iv complete." << endl;

// WriteAction writes scene graph to file
FILE * auv_uvw_iv_fp = fopen ("auv_uvw.iv", "w");
writeaction.getOutput()-> setFilePointer (auv_uvw_iv_fp);
writeaction.apply (root);
fclose (auv_uvw_iv_fp);
cout << " writeaction.apply (root) => auv_uvw.iv complete." << endl;

```

```

FILE * auv_wrl_fp = fopen ("auv.wrl", "w");
writeaction.getOutput()-> setFilePointer (auv_wrl_fp);
writeaction.getOutput()-> setHeaderString ("#VRML V1.0 ascii");
writeaction.apply (AUV_node);
fclose (auv_wrl_fp);
cout << "writeaction.apply (AUV_node) => auv.wrl complete." << endl;

FILE * auv_uvw_wrl_fp = fopen ("auv_uvw.wrl", "w");
writeaction.getOutput()-> setFilePointer (auv_uvw_wrl_fp);
writeaction.getOutput()-> setHeaderString ("#VRML V1.0 ascii");
writeaction.apply (root);
fclose (auv_uvw_wrl_fp);
cout << " writeaction.apply (root) => auv_uvw.wrl complete." << endl;

system ("ivfix -Va auv.iv auv.fix.iv");
cout << "ivfix -Va auv.iv auv.fix.iv complete." << endl;

DIS_net_open ();
atexit (DIS_net_close); // ensure port is reclosed on exit. tested sat.
current_clock = clock (); // initialize

// A TimerSensor updates the object with DIS postures and performs redraws
SoTransform * dummy_xform = new SoTransform;
SoTimerSensor * DIS_Redraw_Sensor = new SoTimerSensor( DIS_Redraw_Callback,
                                                         dummy_xform );

DIS_Redraw_Sensor->setInterval ( 0.10 ); // seconds
DIS_Redraw_Sensor->schedule ();

// system ("rm      sounds/nps_auv.au");
// system ("www -o sounds/nps_auv.au file://taurus.cs.nps.navy.mil/pub/auv/
nps_auv.au");
// system ("www -o sounds/nps_auv.au http://www_tios.cs.utwente.nl/say/
?Naval+Postgraduate+School,Autonomous+Underwater+Vehicle");
// system ("sfplay sounds/nps_auv.au &");

// Ocean surface (surf) wiredraw grid rectangles using FaceSet - - - - -
- - -
// follows AUV graphics object so that it is not culled from scene
const int N = 10;

// type changed from long to int32_t for Inventor 2.1 upgrade
static int32_t numbersurfvertices [(2*N+1)] = {4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
4, 4, 4, 4, 4, 4, 4, 4, 4, 4}; // ref. p. 103

// 2N+1 quadrilaterals, 3 coordinate values per point
static float surfquadvertices [(2*N+1)*4][3];
static int i = 0;
static int j = 0;

for (i = 1; i <= N-1; i = i + 2) // (box pattern black one)
{
    surfquadvertices [j][0] = i; // x
    surfquadvertices [j][1] = 0; // y
    surfquadvertices [j][2] = 0; // z

    surfquadvertices [j+1][0] = N; // x
    surfquadvertices [j+1][1] = N-i; // y
    surfquadvertices [j+1][2] = 0; // z

    surfquadvertices [j+2][0] = N; // x
    surfquadvertices [j+2][1] = N-i+1; // y
    surfquadvertices [j+2][2] = 0; // z

    surfquadvertices [j+3][0] = i-1; // x
    surfquadvertices [j+3][1] = 0; // y
    surfquadvertices [j+3][2] = 0; // z
    j = j + 4;
}

```

```

    }
    if (TRACE)
        cout << "surfquadvertices block 1 complete, j range = [0, " << j-1
            << "]" << endl;

    int old_j = j; // do not modify j, it will continue

    for (i = 0; i <= N-1; i += 2) // (box pattern black two)
    {
        surfquadvertices [j][0] = 0; // x
        surfquadvertices [j][1] = i; // y
        surfquadvertices [j][2] = 0; // z

        surfquadvertices [j+1][0] = N-i; // x
        surfquadvertices [j+1][1] = N; // y
        surfquadvertices [j+1][2] = 0; // z

        surfquadvertices [j+2][0] = N-i-1; // x
        surfquadvertices [j+2][1] = N; // y
        surfquadvertices [j+2][2] = 0; // z

        surfquadvertices [j+3][0] = 0; // x
        surfquadvertices [j+3][1] = i+1; // y
        surfquadvertices [j+3][2] = 0; // z
        j = j + 4;
    }
    if (TRACE)
        cout << "surfquadvertices block 2 complete, j range = [" << old_j << ", "
            << j-1 << "]" << endl;

    for (j = (1*N)*4; j < (2*N)*4; j++) // duplicate, transpose to get second
half
    {
        surfquadvertices [j][0] = surfquadvertices [j - (1*N)*4][0]; //x<-x
        surfquadvertices [j][1] = N-surfquadvertices[j-(1*N)*4][1]; //y<-(10-y)
        surfquadvertices [j][2] = -0.05; // z
    }

    // quadrilaterals now have opposite normal, switch coordinates

    for (i = 40; i < 80; i += 4) // (box pattern red)
    {
        swap_float(surfquadvertices[i][0], surfquadvertices[i+3][0]);
        //points 1,4 x
        swap_float(surfquadvertices[i][1], surfquadvertices[i+3][1]);
        //points 1,4 y
        // z values identical
        swap_float (surfquadvertices [i+1][0], surfquadvertices [i+2][0]);
        // points 2,3 x
        swap_float (surfquadvertices [i+1][1], surfquadvertices [i+2][1]);
        // points 2,3 y
    }

    if (TRACE)
        cout << "surface quads second half duplicate & transpose complete, "
            << "j range = [" << (1*N)*4 << ", " << (2*N)*4 - 1 << "]" << endl;

    // outline square last, must start adjacent to last point to avoid artifact
    j = (2*N)*4;
    {
        surfquadvertices [j][0] = N; // x
        surfquadvertices [j][1] = 0; // y
        surfquadvertices [j][2] = 0; // z

        surfquadvertices [j+1][0] = N; // x
        surfquadvertices [j+1][1] = N; // y
        surfquadvertices [j+1][2] = 0; // z
    }

```

```

    surfquadvertices [j+2][0] = 0;      // x
    surfquadvertices [j+2][1] = N;      // y
    surfquadvertices [j+2][2] = 0;      // z

    surfquadvertices [j+3][0] = 0;      // x
    surfquadvertices [j+3][1] = 0;      // y
    surfquadvertices [j+3][2] = 0;      // z
    j = j + 4;
}
if (TRACE)
    cout << "surfquadvertices outline square block complete, j range = ["
        << (2*N)*4 << ", " << j-1 << "]" << endl;

// use consistent definitions with terrain box <<<
const int BOXSCALE = 40.0;
const int OFFSET = 200.0;
for (j = 0; j < (2*N+1)*4; j++)        // scale and translate all quads
{
    surfquadvertices[j][0] = surfquadvertices [j][0]*BOXSCALE - OFFSET; // x
    surfquadvertices[j][1] = surfquadvertices [j][1]*BOXSCALE - OFFSET; // y
}
if (TRACE)
    cout << "surface quads rescale and transformation complete." << endl;

// Define coordinates for quad vertices & SoFaceSet
SoCoordinate3 *surfcoord = new SoCoordinate3;
surfcoord->point.setValues (0, (2*N+1)*4, surfquadvertices);
cout << "surfcoord->point.setValues(0, " << (2*N+1)*4
    << ", surfquadvertices) complete." << endl;
SoFaceSet *surfquadset = new SoFaceSet;
surfquadset->numVertices.setValues (0, 21, numbersurfvertices);
cout << "surfquadset->numVertices.setValues complete." << endl;
SoComplexity * surfaceComplexity = new SoComplexity;
surfaceComplexity->value = 1.0;

// bring together all the surface components
SoSeparator *surfsection = new SoSeparator;
surfsection->addChild( surfaceComplexity );
if (MINEFIELD)
{
    surfsection->addChild( MineFieldTransform );
    surfsection->addChild( MineFieldScale );
}
surfsection->addChild( wires );
surfsection->addChild( seasurface );
surfsection->addChild( surfcoord );
surfsection->addChild( surfquadset );
root->addChild( surfsection );
cout << "surface wire grid complete." << endl;
// End Ocean surface wiredraw rectangles using FaceSet - - - - -

if (PRINTDIALOG == TRUE)
{
    // Print dialog widget: Inventor training manual p. 9-9
    SoXtPrintDialog *printDialog = new SoXtPrintDialog;
    printDialog->setSceneGraph (root);
    printDialog->show ();
    // printDialog->setWYSIWYG ( TRUE );
    // printDialog->setComponents ( SoOffscreenRenderer::RGB );
}

// Uncomment which viewer you want to use:
SoXtExaminerViewer * viewer = new SoXtExaminerViewer;
// SoXtFlyViewer      * viewer = new SoXtFlyViewer;
// SoXtPlaneViewer    * viewer = new SoXtPlaneViewer;
// SoXtWalkViewer     * viewer = new SoXtWalkViewer;

SbColor background_bluecolor ( 0.1, 0.3, 0.5 );

```



```

SbColor background_blackcolor ( 0.0, 0.0, 0.0 );
SbColor background_greycolor ( 0.1, 0.1, 0.1 );
SbColor background_whitecolor ( 1.0, 1.0, 1.0 );

if (MINEFIELD) viewer->setBackgroundColor( background_greycolor );
else          viewer->setBackgroundColor( background_bluecolor );

/* coloreditor not found?! superceded by SoXtMaterialEditor?
if (BACKGROUNDCOLORDIALOG == TRUE)
{
    // Build the color editor in its own window
    SoXtColorEditor *color_editor = new SoXtColorEditor;
    color_editor->build();
    color_editor->setTitle( "AUV viewer background color" );
    // Add a callback for when the color changes
    color_editor->addColorChangedCallback( colorEditorCB, // the callback
                                          viewer ); // user data to be passed
    color_editor->setColor ( backbluecolor );
    color_editor->show(); // Display the color editor
}
*/
char window_title [60];
strcpy (window_title, "NPS AUV Virtual World (");
strcat (window_title, "multicast address ");
strcat (window_title, group);
strcat (window_title, ", ");
strcat (window_title, "port ");
strcat (window_title, port);
strcat (window_title, ")");

viewer->setSceneGraph( root );
viewer->setTitle(window_title);
viewer->setDecoration (0);
viewer->show();
// XtRealizeWidget ( ViewerWindowWidget ); // mini window junk
SoXt::mainLoop(); // loop forever, sending events to the scene graph

}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

// Read Circles reads from a file containing circle descriptions and adds
those
// circles to the scene graph
void readCircles ()
{
    double circleX      = 0.0;
    double circleY      = 0.0;
    double circleZ      = 0.0;
    double circleRadius = 0.0;
    char circleFlag[50];
    SoSeparator * allCirclesSeparator;
    SoSeparator * circleSeparator;
    SoTranslation * circleTranslate;
    SoRotationXYZ * circleRotation = new SoRotationXYZ;
    SoSeparator * minePairSeparator;
    SoSeparator * mineSeparator;
    SoTranslation * mineTranslate;

    circleRotation->angle = M_PI/2;
    circleRotation->axis = SoRotationXYZ::X;
    SoMaterial * circleMaterial = new SoMaterial;
    circleMaterial->ambientColor.setValue (1.0,0.0,0.0);
    circleMaterial->diffuseColor.setValue (1.0,0.0,0.0);
    // circleMaterial->emissiveColor.setValue(1.0,0.0,0.0); // kills shading
    // transparency > 0.5 shifts to SREENDOOR and looks horrible :(
    circleMaterial->transparency.setValue(0.5);
    circleMaterial->shininess.setValue(1.0);
    SoCylinder * circleCylinder;

```

```

SoPickStyle * pickablestylenode;
pickablestylenode = new SoPickStyle;
pickablestylenode->style.setValue ( SoPickStyle::SHAPE );
SoComplexity * cylinderComplexity = new SoComplexity;
cylinderComplexity->value = 0.3;

allCirclesSeparator = new SoSeparator;
scenery->addChild ( allCirclesSeparator );
allCirclesSeparator->addChild ( cylinderComplexity );
allCirclesSeparator->addChild ( circleMaterial );
allCirclesSeparator->addChild ( pickablestylenode );

char circleFileLine [80];
for (int index = 0; index < 80; index++) circleFileLine [index] = '\0';

while (circleFile.getline (circleFileLine, 80))
{
    if (TRUE) cout << "[" << circleFileLine << "]";
    int paramcount = sscanf (circleFileLine, "%s %lf %lf %lf %lf",
        &circleFlag, &circleX, &circleY, &circleZ, &circleRadius);
    for (index = 0; index < 80; index++) circleFileLine [index] = '\0';

    if (TRUE)
    {
        cout << endl;
        cout << "[parsed from circle file: "
            << circleFlag << " " << circleX << " " << circleY << " "
            << circleZ << " " << circleRadius << "]" << endl;
    }

    for (int index = 0; index <= strlen (circleFlag); index++) //uppercase
        circleFlag [index] = toupper (circleFlag [index]);

    if (((strcmp (circleFlag,"CIRCLE") == 0) ||
        (strcmp (circleFlag,"CYLINDER") == 0) ) &&
        (paramcount == 5) && (circleRadius > 0.0))
    {
        circleSeparator = new SoSeparator;
        circleTranslate = new SoTranslation;
        circleCylinder = new SoCylinder;
        circleCylinder->parts = SoCylinder::SIDES; // let's see through!
        allCirclesSeparator->addChild ( circleSeparator );
        circleTranslate->translation.setValue(circleX,-circleY,-circleZ);
        circleSeparator->addChild ( circleTranslate );
        circleSeparator->addChild ( circleRotation );
        circleCylinder->radius.setValue(circleRadius);
        circleCylinder->height.setValue(2.0 * circleZ);
        circleSeparator->addChild ( circleCylinder );
    }
    else if ((strcmp (circleFlag,"MINE") == 0) &&
        (paramcount >= 4))
    {
        minePairSeparator = new SoSeparator;
        allCirclesSeparator->addChild ( minePairSeparator );
        mineTranslate = new SoTranslation;
        mineTranslate->translation.setValue(circleX,-circleY,-circleZ);
        minePairSeparator->addChild ( mineTranslate );
        mineSeparator = readFile ("mine_spiked.iv");
        minePairSeparator->addChild ( mineSeparator );
    }
    else if (TRUE) cout << "[failed parse test]" << endl;
} // end

//////////////////////////////////////

// Sonar Range Function. Temporary Addition for Testing, moved to dynamics
double auv_ST725_range ( double sonarBearing, double power, double maxSonar-

```

```

Range,
                                SbViewportRegion * pickViewPort )
{
    power = 0.0;  // unused

    double sinPhi  = sin (AUV_phi),
           cosPhi  = cos (AUV_phi),
           sinTheta = sin (AUV_theta),
           cosTheta = cos (AUV_theta),
           sinPsi   = sin (AUV_psi),
           cosPsi   = cos (AUV_psi);

    // SbViewportRegion pickViewPort;

    // Positions of the start and end of the ray in AUV Coordinates
    float beamOffsetX = AUV_ST725_x_offset / 12.0,
          beamOffsetY = AUV_ST725_y_offset / 12.0,
          beamOffsetZ = AUV_ST725_z_offset / 12.0,
          beamAxisX   = cos (radians (sonarBearing)), // 3D
          beamAxisY   = sin (radians (sonarBearing));
    // beamAxisZ   = 0; // unchanging

    // Positions of the start and end of the ray in World Coordinates

    float worldBeamOriginX = beamOffsetX * (cosPsi * cosTheta) +
                              beamOffsetY * (cosPsi * sinTheta * sinPhi -
                                              sinPsi * cosPhi) +
                              beamOffsetZ * (cosPsi * sinTheta * cosPhi +
                                              sinPsi * sinPhi) +
                              AUV_x,
          worldBeamOriginY = beamOffsetX * (sinPsi * cosTheta) +
                              beamOffsetY * (sinPsi * sinTheta * sinPhi +
                                              cosPsi * cosPhi) +
                              beamOffsetZ * (sinPsi * sinTheta * cosPhi -
                                              cosPsi * sinPhi) +
                              AUV_y,
          worldBeamOriginZ = beamOffsetX * (-sinTheta) +
                              beamOffsetY * (cosTheta * sinPhi) +
                              beamOffsetZ * (cosTheta * cosPhi) +
                              AUV_z;

    float worldBeamAxisX = beamAxisX * (cosPsi * cosTheta) +
                            beamAxisY * (cosPsi * sinTheta * sinPhi -
                                          sinPsi * cosPhi),
          worldBeamAxisY = beamAxisX * (sinPsi * cosTheta) +
                            beamAxisY * (sinPsi * sinTheta * sinPhi +
                                          cosPsi * cosPhi),
          worldBeamAxisZ = beamAxisX * (-sinTheta) +
                            beamAxisY * (cosTheta * sinPhi);

    double range = 0.0;

    SoRayPickAction pingPickAction (*pickViewPort);
    pingPickAction.setRay(SbVec3f(worldBeamOriginX,-worldBeamOriginY,-world-
    BeamOriginZ),
                          SbVec3f(worldBeamAxisX,-worldBeamAxisY,-worldBeamAx-
    isZ),
                          0.0,maxSonarRange);

    pingPickAction.apply ( scenery );

    const SoPickedPoint *pingPickedPoint = pingPickAction.getPickedPoint ();
    SbVec3f pingPoint;

    cout << "[ST725 Position " << worldBeamOriginX << " " << worldBeamOriginY
    << " " << worldBeamOriginZ << "]" << endl;
    cout << "[ST725 Beam Axis " << worldBeamAxisX << " " << worldBeamAxisY
    << " " << worldBeamAxisZ << "]" << endl;

```

```

    if (pingPickedPoint != NULL)
    {
        pingPoint = pingPickedPoint->getPoint ();
        cout << "[ST725 ping picked point " << pingPoint[0] << " "
              << -pingPoint[1] << " " << -pingPoint[2] << "]" << endl;
    }
    else
    {
        cout << "[ST725 ping function did not pick any points]" << endl;
        pingPoint = SbVec3f (worldBeamOriginX,-worldBeamOriginY,-worldBeamOriginZ);
    }

    range = sqrt (((worldBeamOriginX - pingPoint[0]) *
                    (worldBeamOriginX - pingPoint[0])) +
                  ((worldBeamOriginY - -pingPoint[1]) *
                    (worldBeamOriginY - -pingPoint[1])) +
                  ((worldBeamOriginZ - -pingPoint[2]) *
                    (worldBeamOriginZ - -pingPoint[2])));

    cout << "[ST725 Sonar Range:  " << range << "]" << endl;
    cout << "[End pingST725]" << endl;
    return (range);
} // end auv_ST725_range

// end of viewer.C
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/*
Program:World database geometric sonar model: ivSonar.C
Description:Geometric sonar model for scene graph.
Author:Duane Davis, Don Brutzman
Revised:20 June 96
System:Irix 5.3
Compiler:ANSI C++
Compilation:irix> make ivSonar.o
            irix> CC ivSonar.C -lm -c -g +w
            -c == Produce binaries only, suppressing the link phase.
            +w == Warn about all questionable constructs.

Description:Generic sonar model for Pheonix AUV ST725 and ST1000 sonars
            using Open Inventor ray pick action and viewer scene graph
*/
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

#include "AUVglobals.H"
#include <iostream.h>
#include <fstream.h>
#include <iomanip.h> // must follow iostream.h
#include <string.h>
#include <math.h>
#include <time.h>
#include <getopt.h>

```

```

#include <stdlib.h>
#include <netdb.h>
#include <netinet/in.h>
#include <sys/types.h>

#include <ctype.h>

////////////////////////////////////

#ifdef INVENTOR

#include <X11/Intrinsic.h>
#include <X11/keysym.h>
#include <Xm/Xm.h>
#include <Xm/CascadeBG.h>
#include <Xm/Form.h>
#include <Xm/RowColumn.h>
#include <Xm/PushB.h>
#include <Xm/PushBG.h>
#include <Xm/Separator.h>
#include <Xm/SeparatorG.h>
#include <Xm/ToggleB.h>
#include <Xm/ToggleBG.h>

#include <Inventor/SbBasic.h>
#include <Inventor/SbViewportRegion.h>
#include <Inventor/So.h>
#include <Inventor/SoDB.h>
#include <Inventor/SoInput.h>
#include <Inventor/SoPickedPoint.h>
#include <Inventor/Xt/SoXt.h>
#include <Inventor/Xt/SoXtPrintDialog.h> // see SoOffscreenRender
#include <Inventor/Xt/SoXtRenderArea.h>
#include <Inventor/Xt/SoXtMaterialEditor.h>
#include <Inventor/Xt/viewers/SoXtExaminerViewer.h>

#include <Inventor/actions/SoAction.h>
#include <Inventor/actions/SoCallbackAction.h>
#include <Inventor/actions/SoGLRenderAction.h>
#include <Inventor/actions/SoWriteAction.h>
#include <Inventor/actions/SoPickAction.h>
#include <Inventor/actions/SoRayPickAction.h>

#include <Inventor/engines/SoCalculator.h>
#include <Inventor/engines/SoElapsedTime.h>
#include <Inventor/engines/SoTimeCounter.h>

#include <Inventor/events/SoEvent.h>
#include <Inventor/events/SoKeyboardEvent.h>

#include <Inventor/nodes/SoComplexity.h>
#include <Inventor/nodes/SoCone.h>

#include <Inventor/nodes/SoCoordinate3.h>
#include <Inventor/nodes/SoCube.h>
#include <Inventor/nodes/SoCylinder.h>
#include <Inventor/nodes/SoCone.h>
#include <Inventor/nodes/SoDirectionalLight.h>
#include <Inventor/nodes/SoDrawStyle.h>
#include <Inventor/nodes/SoEventCallback.h>
#include <Inventor/nodes/SoFaceSet.h>
#include <Inventor/nodes/SoGroup.h>
#include <Inventor/nodes/SoMaterial.h>
#include <Inventor/nodes/SoNurbsSurface.h>
#include <Inventor/nodes/SoPerspectiveCamera.h>
#include <Inventor/nodes/SoPickStyle.h>
#include <Inventor/nodes/SoRotationXYZ.h>
#include <Inventor/nodes/SoScale.h>

```

```

#include <Inventor/nodes/SoSelection.h>
#include <Inventor/nodes/SoSeparator.h>
#include <Inventor/nodes/SoSphere.h>
#include <Inventor/nodes/SoSwitch.h>
#include <Inventor/nodes/SoTransform.h>
#include <Inventor/nodes/SoTransformSeparator.h>
#include <Inventor/nodes/SoTranslation.h>
#include <Inventor/nodes/SoTriangleStripSet.h> // order matters here
#include <Inventor/nodes/SoUnits.h>

////////////////////////////////////

SoSeparator * scenery = new SoSeparator; // objects which can be pinged
SoSeparator * readFile(const char *filename); // Inventor Mentor p. 284
SbViewportRegion pickViewPort; // required for pick functions to work
void readCircles ();

////////////////////////////////////

double radians (double);

extern int READCIRCLEFILE;
extern int LOADDEFAULTWORLD;

extern ifstream circleFile;

////////////////////////////////////

void
setupInventor (char * argument)
{
    // Initialize Inventor and Xt - these steps MUST be first calls
    // in main, without exception, or a mystery crash results.

    Widget ViewerWindowWidget = SoXt::init(argument);
    if ( ViewerWindowWidget == NULL )
    {
        cout << "viewer: ViewerWindowWidget == NULL on startup, exiting."
              << endl;
        exit( 1 );
    }
    return;
} // end setupInventor

////////////////////////////////////

void
loadSceneGraph ()
{
    static int sceneCreated = 0;

    if (sceneCreated) return; // don't load scene more than once
    if (TRACE) cout << "[Begin loadSceneGraph]" << endl;
    sceneCreated = 1; // Don't allow scene to be created again
    scenery->ref();

    if (LOADDEFAULTWORLD)
    {
        // All objects in the scenery graph are pickable

```

```

SoPickStyle * pickablestylenode = new SoPickStyle;
pickablestylenode->style.setValue ( SoPickStyle::SHAPE );
scenery->addChild(pickablestylenode);

// Platform
SoSeparator * sepPlatform = new SoSeparator;
SoTransform * xfPlatform = new SoTransform;
SoSeparator * Platform = readFile ("../viewer/Platform.iv");
xfPlatform->translation.setValue(-80.0, 50.0, -50.0);
sepPlatform->addChild ( xfPlatform );
sepPlatform->addChild( Platform );
scenery->addChild ( sepPlatform );

// Testtank
SoSeparator * sepTesttank = new SoSeparator;
SoTransform * xfTesttank = new SoTransform;
SoSeparator * Testtank = readFile ("../viewer/testtank.iv");
if (TESTTANKSURFACE)
    xfTesttank->translation.setValue( 0.0, 0.0, -6.0);
else
    xfTesttank->translation.setValue( 0.0, 0.0, -50.0);
sepTesttank->addChild ( xfTesttank );
sepTesttank->addChild( Testtank );
scenery->addChild ( sepTesttank );

// Torpedo Tube
SoSeparator * sepTorpedoTube = new SoSeparator;
SoTransform * xfTorpedoTube = new SoTransform;
xfTorpedoTube->translation.setValue( 30.0, 20.0, -48.00 );
SoCylinder * TorpedoTube = new SoCylinder;
TorpedoTube->radius = 21.0 / 12.0; // 21" diameter
TorpedoTube->height = 10.0; // 10' length
TorpedoTube->parts = SoCylinder::SIDES; // let's drive through!
sepTorpedoTube->addChild ( xfTorpedoTube );
sepTorpedoTube->addChild( TorpedoTube );
scenery->addChild ( sepTorpedoTube );

SoCube * backgroundCube = new SoCube;
backgroundCube->width = 400.0;
backgroundCube->height = 400.0;
backgroundCube->depth = 0.1;
SoTransform * xfbackgroundCube = new SoTransform;
xfbackgroundCube->translation.setValue(0.0, 0.0, -50.0);
SoSeparator * sepbackgroundCube = new SoSeparator;
sepbackgroundCube->addChild( xfbackgroundCube );
sepbackgroundCube->addChild( backgroundCube ); // remove cube here
scenery->addChild ( sepbackgroundCube );
}

// WriteAction writes scene graph to file
FILE * scenery_iv_fp = fopen ("dynamics_scenery.iv", "w");
SoWriteAction writeaction;
writeaction.getOutput()-> setFilePointer (scenery_iv_fp);
writeaction.apply (scenery);
fclose (scenery_iv_fp);
cout << " writeaction.apply (scenery) => dynamics_scenery.iv complete."
    << endl;

if (TRACE) cout << "[End loadSceneGraph]" << endl;
} // end loadSceneGraph

////////////////////////////////////
double pingST725 ( double maxSonarRange, double powerSetting )
{
// int TRACE = 1;

```

```

if (TRACE) cout << "[Begin pingST725]" << endl;

int sonarBin[64] = {0};
int bin;
int bestBin = 0;

double binSize = maxSonarRange / 64.0;

SoPickedPoint *pingPickedPoint;
SbVec3f pingPoint;

double range = 0.0;

double sinPhi   = sin (AUV_phi),
      cosPhi    = cos (AUV_phi),
      sinTheta  = sin (AUV_theta),
      cosTheta  = cos (AUV_theta),
      sinPsi    = sin (AUV_psi),
      cosPsi    = cos (AUV_psi);

powerSetting = 10.0; // currently unused

// Positions of the start and end of the ray in AUV Coordinates
// Beam Axis is for center of 24 degree vertical arc
// Beam Axis must be recomputed for top and bottom of beam
float beamOffsetX = AUV_ST725_x_offset / 12.0,
      beamOffsetY = AUV_ST725_y_offset / 12.0,
      beamOffsetZ = AUV_ST725_z_offset / 12.0,
      beamAxisX   = cos (radians (AUV_ST725_bearing)),
      beamAxisY   = sin (radians (AUV_ST725_bearing)),
      beamAxisZ   = 0;

// Positions of the start and end of the ray in World Coordinates

float worldBeamOriginX = beamOffsetX * (cosPsi * cosTheta) +
      beamOffsetY * (cosPsi * sinTheta * sinPhi -
      sinPsi * cosPhi) +
      beamOffsetZ * (cosPsi * sinTheta * cosPhi +
      sinPsi * sinPhi) +
      AUV_x,
      worldBeamOriginY = beamOffsetX * (sinPsi * cosTheta) +
      beamOffsetY * (sinPsi * sinTheta * sinPhi +
      cosPsi * cosPhi) +
      beamOffsetZ * (sinPsi * sinTheta * cosPhi -
      cosPsi * sinPhi) +
      AUV_y,
      worldBeamOriginZ = beamOffsetX * (-sinTheta) +
      beamOffsetY * (cosTheta * sinPhi) +
      beamOffsetZ * (cosTheta * cosPhi) +
      AUV_z;

float worldBeamAxisX,
      worldBeamAxisY,
      worldBeamAxisZ;

SoRayPickAction pingPickAction (pickViewPort);

for (double rayElevation = -12.0; rayElevation <= 12.0; rayElevation += 2.0)
{
    beamAxisZ = tan ( rayElevation * M_PI / 180.0);

    worldBeamAxisX = beamAxisX * (cosPsi * cosTheta) +
        beamAxisY * (cosPsi * sinTheta * sinPhi -
        sinPsi * cosPhi) +
        beamAxisZ * (cosPsi * sinTheta * cosPhi +
        sinPsi * sinPhi),
        worldBeamAxisY = beamAxisX * (sinPsi * cosTheta) +

```



```

        beamAxisY * (sinPsi * sinTheta * sinPhi +
        cosPsi * cosPhi) +
        beamAxisZ * (sinPsi * sinTheta * cosPhi -
        cosPsi * sinPhi),
worldBeamAxisZ = beamAxisX * (-sinTheta) +
        beamAxisY * (cosTheta * sinPhi) +
        beamAxisZ * (cosTheta * cosPhi);

// Perform pick
pingPickAction.setRay(SbVec3f(worldBeamOriginX,
        -worldBeamOriginY,
        -worldBeamOriginZ),
        SbVec3f(worldBeamAxisX,
        -worldBeamAxisY,
        -worldBeamAxisZ),
        0.0,maxSonarRange);

if (TRACE)
{
    cout << "[ST725 Position " << worldBeamOriginX << " "
    << worldBeamOriginY
    << " " << worldBeamOriginZ << "]" << endl;
    cout << "[ST725 Beam Axis " << worldBeamAxisX << " "
    << worldBeamAxisY
    << " " << worldBeamAxisZ << "]" << endl;
}

pingPickAction.apply ( scenery );
pingPickedPoint = pingPickAction.getPickedPoint ();

if (pingPickedPoint != NULL)
{
    pingPoint = pingPickedPoint->getPoint ();
    if (TRACE)
        cout << "[ST725 " << rayElevation << " Degree Ray Picked Point"
        << pingPoint[0] << " "
        << -pingPoint[1] << " " << -pingPoint[2] << "]" << endl;
    range = sqrt (((worldBeamOriginX - pingPoint[0]) *
        (worldBeamOriginX - pingPoint[0])) +
        ((worldBeamOriginY + pingPoint[1]) *
        (worldBeamOriginY + pingPoint[1])) +
        ((worldBeamOriginZ + pingPoint[2]) *
        (worldBeamOriginZ + pingPoint[2])));
}
else
{
    if (TRACE)
        cout << "[ST725 " << rayElevation
        << " degree ray did not Pick any Points]" << endl;
    range = 10000.0; // a big number that will exceed max sonar range
}

if (TRACE) cout << "[ST725 " << rayElevation << " degree range: "
    << range << "]" << endl;

bin = (int) (range / binSize);

if (bin < 64)
{
    if (TRACE) cout << "[Incrementing bin number: " << bin << "]"
        << endl;
    ++sonarBin[bin];
}
}

// Determine range to return

```

```

for (int i = 0; i < 64; i++)
{
    if (sonarBin[i] > sonarBin[bestBin])
        bestBin = i;
}

if (sonarBin[bestBin] > 1)
{
    // range_error: +/- (0..1) * max_percentage_error * sonar_return_value

    double sonar_return_value = (binSize * bestBin + binSize / 2.0);
    double range_error = sonar_return_value *
        (max_pct_error_ST725 / 100.0) *
        (2.0 * (drand48 () - 0.5));

    if (TRACE) cout << "ST_725 range = " << sonar_return_value << ", ";
    if (TRACE) cout << "range error = " << range_error << ", ";

    sonar_return_value += range_error;

    if (TRACE) cout << "error return = " << sonar_return_value << endl;

    return (sonar_return_value);
}
else return (0.0);
} // end pingST725

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
double pingST1000 ( double maxSonarRange, double powerSetting )
{
    // int TRACE = 1;

    if (TRACE)
        cout << "[Begin pingST1000]" << endl;

    double sinPhi = sin (AUV_phi),
           cosPhi = cos (AUV_phi),
           sinTheta = sin (AUV_theta),
           cosTheta = cos (AUV_theta),
           sinPsi = sin (AUV_psi),
           cosPsi = cos (AUV_psi);

    powerSetting = 10.0; // currently unused

    // SbViewportRegion pickViewPort;

    // Positions of the start and end of the ray in AUV Coordinates
    float beamOffsetX = AUV_ST1000_x_offset / 12.0,
          beamOffsetY = AUV_ST1000_y_offset / 12.0,
          beamOffsetZ = AUV_ST1000_z_offset / 12.0,
          beamAxisX = cos (radians (AUV_ST1000_bearing)),
          beamAxisY = sin (radians (AUV_ST1000_bearing));
    // beamAxisZ = 0; // unchanged

    // Positions of the start and end of the ray in World Coordinates
    float worldBeamOriginX = beamOffsetX * (cosPsi * cosTheta) +
                             beamOffsetY * (cosPsi * sinTheta * sinPhi -
                             sinPsi * cosPhi) +
                             beamOffsetZ * (cosPsi * sinTheta * cosPhi +
                             sinPsi * sinPhi) +
                             AUV_x,
          worldBeamOriginY = beamOffsetX * (sinPsi * cosTheta) +
                             beamOffsetY * (sinPsi * sinTheta * sinPhi +
                             cosPsi * cosPhi) +
                             beamOffsetZ * (sinPsi * sinTheta * cosPhi -

```

```

                                cosPsi * sinPhi) +
worldBeamOriginZ = beamOffsetX * (-sinTheta) +
                    beamOffsetY * (cosTheta * sinPhi) +
                    beamOffsetZ * (cosTheta * cosPhi) +
                    AUV_z;

float worldBeamAxisX = beamAxisX * (cosPsi * cosTheta) +
                      beamAxisY * (cosPsi * sinTheta * sinPhi -
                      sinPsi * cosPhi),
worldBeamAxisY = beamAxisX * (sinPsi * cosTheta) +
                  beamAxisY * (sinPsi * sinTheta * sinPhi +
                  cosPsi * cosPhi),
worldBeamAxisZ = beamAxisX * (-sinTheta) +
                  beamAxisY * (cosTheta * sinPhi);

double range = 0.0;

SoRayPickAction pingPickAction (pickViewPort);
pingPickAction.setRay(SbVec3f(worldBeamOriginX,
                              -worldBeamOriginY,
                              -worldBeamOriginZ),
                     SbVec3f(worldBeamAxisX,
                              -worldBeamAxisY,
                              -worldBeamAxisZ),
                     0.0,maxSonarRange);

pingPickAction.apply ( scenery );

const SoPickedPoint *pingPickedPoint = pingPickAction.getPickedPoint ();
SbVec3f pingPoint;

if (TRACE)
{
    cout << "[ST1000 Position " << worldBeamOriginX << " "
          << worldBeamOriginY
          << " " << worldBeamOriginZ << "]" << endl;
    cout << "[ST1000 Beam Axis " << worldBeamAxisX << " "
          << worldBeamAxisY
          << " " << worldBeamAxisZ << "]" << endl;
}

if (pingPickedPoint != NULL)
{
    pingPoint = pingPickedPoint->getPoint ();
    if (TRACE)
        cout << "[ST1000 Ping Picked Point " << pingPoint[0] << " "
              << -pingPoint[1] << " " << -pingPoint[2] << "]" << endl;
    }
else
{
    if (TRACE) cout << "[ST1000 Ping Function did not Pick any Points]"
                  << endl;
    pingPoint = SbVec3f (worldBeamOriginX,
                        -worldBeamOriginY,
                        -worldBeamOriginZ);
}

range = sqrt (((worldBeamOriginX - pingPoint[0]) *
                (worldBeamOriginX - pingPoint[0])) +
              ((worldBeamOriginY - -pingPoint[1]) *
                (worldBeamOriginY - -pingPoint[1])) +
              ((worldBeamOriginZ - -pingPoint[2]) *
                (worldBeamOriginZ - -pingPoint[2])));

// range_error: +/- (0..1) * max_percentage_error * sonar_return_value
double sonar_return_value = range;

```

```

double range_error = sonar_return_value *
    (max_pct_error_ST1000 / 100.0) *
    (2.0 * (drand48 () - 0.5));

if (TRACE) cout << "ST1000 range = " << sonar_return_value << ", ";
if (TRACE) cout << "range error = " << range_error << ", ";

sonar_return_value += range_error;

if (TRACE) cout << "error return = " << sonar_return_value << endl;
if (TRACE) cout << "[End pingST1000]" << endl;

return (sonar_return_value);
} // end pingST1000

/////////////////////////////////////////////////////////////////

void
addWorldFile (const char *filename)
{
    SoSeparator * worldSeparator = readFile (filename);
    if (!worldSeparator)
    {
        char newfilename [50];
        strcpy (newfilename, "../viewer/");
        strcat (newfilename, filename);
        worldSeparator = readFile (newfilename);
    }
    if (worldSeparator)
    {
        scenery->addChild (worldSeparator);
        cout << "[World File " << filename << " opened]" << endl;
    }
    else
    {
        cout << "Unable to locate world-file " << filename << endl;
        exit (-1);
    }
}

/////////////////////////////////////////////////////////////////

SoSeparator * readFile(const char *filename) // Inventor Mentor p. 284
{
    // Open the input file
    SoInput mySceneInput;
    if (!mySceneInput.openFile(filename))
    {
        cout << "Cannot open file " << filename << endl;
        return NULL;
    }
    // Read the whole file into the database
    SoSeparator * myGraph = SoDB::readAll(&mySceneInput);
    if (myGraph == NULL)
    {
        cout << "Problem reading file " << filename << endl;
        return NULL;
    }
    mySceneInput.closeFile ();
    return myGraph;
}

/////////////////////////////////////////////////////////////////

// Read Circles reads from a file containing circle descriptions and adds
// those circles to the scene graph

```

```

void readCircles ()
{
    double circleX, circleY, circleZ, circleRadius;
    char circleFlag[50];
    SoSeparator * allCirclesSeparator;
    SoSeparator * circleSeparator;
    SoTranslation * circleTranslate;
    SoSeparator * minePairSeparator;
    SoSeparator * mineSeparator;
    SoTranslation * mineTranslate;
    SoRotationXYZ * circleRotation = new SoRotationXYZ;

    circleRotation->angle = M_PI/2;
    circleRotation->axis = SoRotationXYZ::X;

    SoMaterial * circleMaterial = new SoMaterial;
    // circleMaterial->ambientColor.setValue (1.0,0.0,0.0);
    // circleMaterial->diffuseColor.setValue (1.0,0.0,0.0);
    // circleMaterial->emissiveColor.setValue(1.0,0.0,0.0);
    circleMaterial->shininess.setValue(0.0);

    SoCylinder * circleCylinder;
    SoPickStyle * pickablestyleNode;
    pickablestyleNode = new SoPickStyle;
    pickablestyleNode->style.setValue ( SoPickStyle::SHAPE );
    SoComplexity * cylinderComplexity = new SoComplexity;
    cylinderComplexity->value = 0.4;

    allCirclesSeparator = new SoSeparator;
    scenery->addChild ( allCirclesSeparator );
    allCirclesSeparator->addChild ( cylinderComplexity );
    // allCirclesSeparator->addChild ( circleMaterial );
    allCirclesSeparator->addChild ( pickablestyleNode );

    char circleFileLine [80];
    for (int index = 0; index < 80; index++) circleFileLine [index] = '\0';
    while (circleFile.getLine (circleFileLine, 80))
    {
        if (TRUE) cout << "[" << circleFileLine << "]" << endl;
        int paramcount = sscanf (circleFileLine, "%s %lf %lf %lf %lf",
            &circleFlag, &circleX, &circleY, &circleZ, &circleRadius);
        for (index = 0; index < 80; index++) circleFileLine [index] = '\0';

        if (TRUE)
        {
            cout << "[parsed from circle file: "
                << circleFlag << " " << circleX << " " << circleY << " "
                << circleZ << " " << circleRadius << "]" << endl;
        }

        for (int index = 0; index <= strlen (circleFlag); index++) //uppercase
            circleFlag [index] = toupper (circleFlag [index]);

        if (((strcmp (circleFlag, "CIRCLE") == 0) ||
            (strcmp (circleFlag, "CYLINDER") == 0) ) &&
            (paramcount == 5) && (circleRadius > 0.0))
        {
            circleSeparator = new SoSeparator;
            circleTranslate = new SoTranslation;
            circleCylinder = new SoCylinder;
            circleCylinder->parts = SoCylinder::SIDES; // let's see through!
            allCirclesSeparator->addChild ( circleSeparator );
            circleTranslate->translation.setValue(circleX,-circleY,-circleZ);
            circleSeparator->addChild ( circleTranslate );
            circleSeparator->addChild ( circleRotation );
            circleCylinder->radius.setValue(circleRadius);
            circleCylinder->height.setValue(2.0 * circleZ);
            circleSeparator->addChild ( circleCylinder );
        }
    }
}

```

```

    }
    else if ((strcmp (circleFlag,"MINE") == 0)  &&
              (paramcount >= 4))
    {
        minePairSeparator = new SoSeparator;
        allCirclesSeparator->addChild ( minePairSeparator );
        mineTranslate = new SoTranslation;
        mineTranslate->translation.setValue(circleX,-circleY,-circleZ);
        minePairSeparator->addChild ( mineTranslate );
        mineSeparator = readFile ("../viewer/mine_spiked.iv");
        minePairSeparator->addChild ( mineSeparator );
    }
    else if (TRUE) cout << "[failed parse test]" << endl;
}
} // end readCircles

#endif
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

```

III. EXECUTION LEVEL SOURCE CODE

This section contains source code for the *Phoenix* execution level. Included files are `defines.h`, `statevector.h`, `statevector.c`, `globals.h`, `globals.c`, `external_functions.c`, `parse_functions.c` and `execution.c`. These files are compiled individually and linked together to form a single executable program. These files are available online at http://www.stl.nps.navy.mil/~brutzman/dissertation/software_reference.html

Files are available individually or as part of a .tar package containing all *Phoenix* AUV and underwater virtual world source code and instructions on their installation and use. The execution level files can be compiled and linked to run on the Silicon Graphics workstations using the Makefile available at the above location. To create an executable for the physical vehicle, the PC cross compiler must be used. This compiler requires that `statevector.c`, `globals.c`, `external_functions.c` and `parse_functions.c` be combined into a single file before compilation as follows:

```
> cat globals.c statevector.c external_functions.c parse_functions.c > exf.c
```

The files `defines.h`, `statevector.h`, `globals.h`, `exf.c` and `execution.c` can then be moved to the BURNS directory on the PC and compiled using the `linke.bat` batch file. The resulting executable will be called `executio`.

```

/*****
/*
Program:      defines.h

Authors:      Don Brutzman and Duane Davis

Revised:      2 August 96

System:       AUV Gespac 68020/68030, OS-9 version 2.4
Compiler:     Gespac cc Kernighan & Richie (K&R) C

Purpose:      Allows repeated use of global variables global.c via global.h
               with one set of defines in order to prevent compiler warnings

               Some massive cleanup will someday be needed here!

*/

/*****

#ifndef DEFINES_H
#define DEFINES_H

/*****

/* Figure out architecture.  Insert an error to verify right choice fires. */
#define os9
#ifdef sgi
#undef os9
#endif
#ifdef sun
#undef os9
#endif

/*****

/* Defined for Iris and OS-9, located in the same subdirectories      */

#include <ctype.h>
#include <math.h>
#include <time.h>

/* Unix defines first */

#ifndef os9

/* Irix defines in /usr/include */
/* sun4 defines in /usr/include */

/* defined for Iris and OS-9 but in different places */

#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <signal.h>
#include <string.h>

/* Unix library version */
#include <errno.h>

#include "modes.h"
#include "setsys.h"
#include "sgstat.h"

```



```

/* Now OS-9 subdirectories /DEFS, /DEFS/INET */
#else

#include <stdio.h>
/* defined for Iris and OS-9 but in different places */
#include <types.h>
#include <inet/socket.h>
#include <inet/in.h>
#include <inet/netdb.h>
#include <signal.h>

/* OS-9 local version (put it there) */
#include "errno.h"

#include <modes.h>
/* #include <setsys.h> */
#include <sgstat.h>

/* see /h0/DEFS/time.h */
#define CLOCKS_PER_SEC CLK_TCK

#endif

/* CAUTION: the order of the above files is very finicky & can cause errors */
/* if changed. Also note additional dependencies in OS-9 makefile. */

/*****
/* files and paths */

#define AUVINFOFILENAME "mission.info"
#define AUVINITFILENAME "mission.init"
#define AUVSCRIPTFILENAME "mission.script"

#ifdef os9
#define AUVORDERSFILENAME "mission.output.orders"
#define AUVDATAFILENAME "mission.output.telemetry"
#define AUVTEXTFILENAME "mission.output.1_second"
#define ST1000DATAFILE "st1000datafile"
#define TARGETDATAFILE "targetdatafile"
#else
#define AUVORDERSFILENAME "/r1/mission.output.orders"
#define AUVDATAFILENAME "/r1/mission.output.telemetry"
#define AUVTEXTFILENAME "/r1/mission.output.1_second"
#define ST1000DATAFILE "/r1/st1000datafile"
#define TARGETDATAFILE "/r1/targetdatafile"
#endif
#define AUVEMAILFILENAME "mission.output.email"
#define CONTROLCONSTANTSINPUTNAME "control.constants.input"
#define CONTROLCONSTANTSOUTPUTNAME "control.constants.output"

/* not included in public distribution */
#define EMAILADDRESSFILENAME "mission_email_addresses"

/*****
/* function definitions */

#define sqr(x) x*x

/*****
/* constant definitions */

#define TRUE 1
#define FALSE 0

#define ON 1
#define OFF 0

```

```

#define YES      1
#define NO       0

#define LEFT -1
#define RIGHT 1

/* OS-9 has PI in <math.h> */

#ifndef PI
#define PI      3.1415926535897932
#endif

/* note:  these pointer constants should be cast to the right type when used */

#define DAC2B_ADDR      0xFFF00040/* updated */
#define DAC_LSB_OFFSET  0x2/* updated */
#define DAC1_ADDR      0xFFF00000/* updated */

#define ADC1_ADDR      (DAC1_ADDR + 0x11)/* updated */
#define ADC1_MSB      0x0/* updated */
#define ADC1_LSB      0x2/* updated */
#define ADC1_CMD_REG   0x4/* updated */
#define ADC1_STATUS_REG 0x4/* updated */
#define ADC1_BUSY      0x40/* updated */

#define ADC2_ADDR      0xFFF00020/* updated */
#define ADC2_CH_GAIN   0x0/* updated */
#define ADC2_STATUS_REG 0x2/* updated */
#define ADC2_DATA      0x1/* updated */
#define ADC2_CMD_REG   0x2/* updated */

/* watch out for potential conflict between DACS & input port /P !!!!!!! */

/* next 2: updated */
#define VIA0_ADDR 0xFFF00080 /* PIA Card Base Address VIA0 Port */
#define VIA1_ADDR(VIA0_ADDR + 0x20)/* VIA1 Port */

#define ORB_IRB      1/* updated */
#define ORA_IRA      3/* updated */
#define DDRB         5/* updated */
#define DDRA         7/* updated */

#define T1C_L        4
#define T1C_H        5
#define T1L_L        6
#define T1L_H        7
#define T2C_L        8
#define T2C_H        9
#define ACR          11
#define PCR          12
#define IFR          13
#define IER          14
#define SONAR_SW1     0x0E
#define SONAR_SW2     0x0D
#define SONAR_SW3     0x0B
#define SONAR_SW4     0x07
#define SONAR_TRIG1   0x10
#define SONAR_TRIG2   0x20

#define AVG_PTS       10
#define MAX_RNG_DIFF  22.0
#define MIN_NO_PTS    3
#define MAX_BAD_PTS   3

/* verification required */
#define BOW_RUDDER_TOP      1
#define BOW_RUDDER_BOTTOM  1

```

```

#define BOW_PLANE_STBD      3
#define BOW_PLANE_PORT      3
#define STERN_RUDDER_TOP    2
#define STERN_RUDDER_BOTTOM 2
#define STERN_PLANE_STBD    4
#define STERN_PLANE_PORT    4

#define CONVERT_TO_FEET     0.02398

#define RIGHT_MOTOR         0
#define LEFT_MOTOR          2
#define SUPPLY               1

#define RIGHT_MOTOR_RPM     0
#define LEFT_MOTOR_RPM      1

/* For ADC2 Card */
#define ROLL_ANGLE_CH       12/* updated */
#define PITCH_ANGLE_CH      11/* updated */
#define ROLL_RATE_CH        9 /* updated */
#define PITCH_RATE_CH       8/* updated */
#define YAW_RATE_CH         10/* updated */
#define DEPTH_CELL_CH       7/* updated */
#define DOWN_SONAR_CH       14/* updated */

/* For ADA1 Card */
#define COMPUTER_VOLTAGE_CH  7/* updated */
#define MOTOR_GYRO_VOLTAGE_CH 8/* updated */

#define MFI_BASE (0xFFFF00700)/* updated */
                                /* parallel port defines are from mfi_a3.c */
                                /* this is the board base address on the G96 bus */
#define MFI_INPUT_PORT      0
#define MFI_OUTPUT_PORT     1

/* One stream socket is used with adequate throughput */
/* (although two could work, no performance improvement is expected) */

/* Be careful that you reserve these port numbers to prevent collisions */
/* from other processes requesting ports on your system: */

#define DISBRIDGE_TCP_PORT   2056 /* disbridge 1.3 program, server & client */
#define NPSNET_MCAST_PORT    3111 /* Mike Macedonia's multicast DIS 2.0.3 */

                                /* NPS Autonomous Underwater Vehicle (AUV) */
                                /* Underwater Virtual World (UVW) */

#define AUVSIM1_TCP_PORT_0   3210 /* os9sender <==> os9server test programs */
#define AUVSIM1_TCP_PORT_1   3211 /* auv execution level <==> virtual world */
#define AUVSIM1_TCP_PORT_2   3212 /* auv execution <==> tactical (teleme-try) */
#define AUVSIM1_TCP_PORT_3   3213 /* auv execution <==> tactical (messages) */
#define AUVSIM1_TCP_PORT_4   3214 /* port for future use */
#define AUVSIM1_TCP_PORT_5   3215 /* port for future use */
#define AUVSIM1_TCP_PORT_6   3216 /* port for future use */
#define AUVSIM1_TCP_PORT_7   3217 /* port for future use */
#define AUVSIM1_TCP_PORT_8   3218 /* port for future use */

```



```

/*****
/*
Program:          statevector.h

                  State vector (telemetry variables) common definition

Authors:          Don Brutzman, Mike Burns and Duane Davis

Revised:          2 May 96

System:           AUV Gespac 68020/68030, OS-9 version 2.4
Compiler:         Gespac cc Kernighan & Richie (K&R) C

Compilation:      ftp>      put statevector.c
                  auvsim1> chd execution
                  [68020]   auvsim1> make -k2f execution
                  [68030]   auvsim1> make      execution

                  [Irix ]   fletch> make execution

Purpose:          Allows repeated use of global variables in statevector.c
                  via statevector.h in order to prevent compiler warnings

                  See globals.c/globals.h for other global variables

Religion:         All distance units are feet, all time units are seconds,
                  all rotational units are degrees.  This is only required
                  when transmitting values externally (socket/text/file).

                  Deciding factors are consistency and human readability..
                  Computational performance is not an issue.

                  Anyone who disagrees has to put up with an endless argument
                  from Don who will not be persuaded to accept any variations!

*/
/*****

#include "defines.h"

/*****
#ifndef STATEVECTOR_H
#define STATEVECTOR_H
/*****

extern int          STATEVECTORSIZE          ; /* how many variables */
extern char         keyword [300]            ; /* auv_state or uvw_state */

extern double       t                        ; /* units are seconds */
extern double       x                        ; /* feet */
extern double       y                        ; /* feet */
extern double       z                        ; /* feet */
extern double       phi                      ; /* degrees */
extern double       theta                    ; /* degrees */
extern double       psi                      ; /* degrees */
extern double       x_dot                    ; /* feet/sec */
extern double       y_dot                    ; /* feet/sec */
extern double       z_dot                    ; /* feet/sec */
extern double       phi_dot                  ; /* degrees/sec */
extern double       theta_dot                ; /* degrees/sec */
extern double       psi_dot                 ; /* degrees/sec */
extern double       speed                    ; /* feet/sec (paddlewheel) */
                                           /* possibly averaged */
extern double       u                        ; /* feet/sec */
extern double       v                        ; /* feet/sec */
extern double       w                        ; /* feet/sec */

```

```

extern double      p                ; /* degrees/sec */
extern double      q                ; /* degrees/sec */
extern double      r                ; /* degrees/sec */
extern double      delta_planes     ; /* degrees */
extern double      delta_rudder     ; /* degrees */
extern double      port_rpm         ; /* -700..700 */
extern double      stbd_rpm         ; /* -700..700 */

/* +- 24V <=> +-2 lb, + Volts moves thruster in + direction, all identical*/
extern double AUV_bow_vertical ; /* thruster rpm */
extern double AUV_stern_vertical ; /* thruster rpm */
extern double AUV_bow_lateral ; /* thruster rpm */
extern double AUV_stern_lateral ; /* thruster rpm */

/* warning: do not use leading zero with bearings or else read as octal */
extern double AUV_ST1000_bearing ; /* ST_1000 conical bearing degrees*/
extern double AUV_ST1000_range ; /* ST_1000 conical range feet */
extern double AUV_ST1000_strength; /* ST_1000 conical strength dB */

extern double AUV_ST725_bearing ; /* ST_725 1 x 24 sector bearing degrees*/
extern double AUV_ST725_range ; /* ST_725 1 x 24 sector range feet */
extern double AUV_ST725_strength ; /* ST_725 1 x 24 sector strength dB */

extern double divetracker_range1; /* feet range to divetracker unit 1 */
extern double divetracker_range2; /* feet range to divetracker unit 2 */
/* negative range means invalid return */
/* future: divetracker_heading1 & 2 */

/*****/

#endif
/* end statevector.h */

```

```

/*****
/*
Program:          statevector.c

                  State vector (telemetry variables) common definition

Authors:         Don Brutzman, Mike Burns and Duane Davis

Revised:         9 June 96

System:          AUV Gespac 68020/68030, OS-9 version 2.4
Compiler:        Gespac cc Kernighan & Richie (K&R) C

Compilation:     ftp>      put statevector.c
                  auvsim1> chd execution
                  [68020]  auvsim1> make -k2f execution
                  [68030]  auvsim1> make      execution

                  [Irix ]  fletch> make execution

Purpose:         Allows repeated use of global variables in statevector.c
                  via statevector.h in order to prevent compiler warnings

                  See globals.c/globals.h for other global variables

Religion:        All distance units are feet, all time units are seconds,
                  all rotational units are degrees.  This is only required
                  when transmitting values externally (socket/text/file).

                  Deciding factors are consistency and human readability.
                  Computational performance is not an issue.

                  Anyone who disagrees has to put up with an endless argument
                  from Don who will not be persuaded to accept any variations!
*/

```

```

/*****
#include "defines.h"
/*****

int          STATEVECTORSIZE      = 37; /* how many variables follow*/
char         keyword [300];       /* auv_state or uvw_state */

double       t                    = 0.0; /* units are seconds */
double       x                    = 5.0; /* feet */
double       y                    = 5.0; /* feet */
double       z                    = 2.0; /* feet */
double       phi                  = 0.0; /* degrees */
double       theta                = 0.0; /* degrees */
double       psi                  = 0.0; /* degrees */
double       x_dot                = 0.0; /* feet/sec */
double       y_dot                = 0.0; /* feet/sec */
double       z_dot                = 0.0; /* feet/sec */
double       phi_dot              = 0.0; /* degrees/sec */
double       theta_dot            = 0.0; /* degrees/sec */
double       psi_dot              = 0.0; /* degrees/sec */
double       speed                = 0.0; /* feet/sec (paddlewheel) */
                        /* possibly averaged */
double       u                    = 0.0; /* feet/sec */
double       v                    = 0.0; /* feet/sec */
double       w                    = 0.0; /* feet/sec */
double       p                    = 0.0; /* degrees/sec */
double       q                    = 0.0; /* degrees/sec */
double       r                    = 0.0; /* degrees/sec */

```

```

double      delta_planes      = 0.0; /* degrees      */
double      delta_rudder     = 0.0; /* degrees      */
double      port_rpm         = 0 ; /* -700..700    */
double      stbd_rpm         = 0 ; /* -700..700    */

/* +- 24V <=> +-2 lb, + Volts moves thruster in + direction, all identical*/
double AUV_bow_vertical = 0.0; /* thruster rpm */
double AUV_stern_vertical = 0.0; /* thruster rpm */
double AUV_bow_lateral = 0.0; /* thruster rpm */
double AUV_stern_lateral = 0.0; /* thruster rpm */

/* warning: do not use leading zero with bearings or else read as octal */

double AUV_ST1000_bearing = 0.0; /* ST_1000 conical bearing degrees */
double AUV_ST1000_range = 0.0; /* ST_1000 conical range feet */
double AUV_ST1000_strength = -1.0; /* ST_1000 conical strength dB */

double AUV_ST725_bearing = 0.0; /* ST_725 1 x 24 sector bearing degrees */
double AUV_ST725_range = 0.0; /* ST_725 1 x 24 sector range feet */
double AUV_ST725_strength = -1.0; /* ST_725 1 x 24 sector strength dB */

double divetracker_range1 = -1.0; /* feet range to divetracker unit 1 */
double divetracker_range2 = -1.0; /* feet range to divetracker unit 2 */
/* negative range means invalid return */
/* future: divetracker_heading1 & 2 */

/*****
/* end statevector.h */

```



```

/*****/
/*
Program:      globals.h

Authors:      Don Brutzman, Duane Davis

Revised:      2 August 96

System:       AUV Gespac 68020/68030, OS-9 version 2.4
Compiler:     Gespac cc Kernighan & Richie (K&R) C

Compilation:  ftp>      put globals.h
               auvsim1> chd execution
               [68020]  auvsim1> make -k2f execution
               [68030]  auvsim1> make      execution

               [Irix ]  fletch> make execution

Purpose:      Allows repeated use of global variables global.c via global.h
               in order to prevent compiler warnings

               See statevector.c/statevector.h for other global variables

*/

/*****/
#ifndef GLOBALS_H
#define GLOBALS_H
/*****/

#include "defines.h"

#ifdef os9

/* these are marco routines for divetracker */
#include "dt2cl.h"
#include "modulo.h"

#endif

/*****/
/* Program configuration flags */
extern int    TRACE                ; /* 1=trace on,      0=trace off      */
extern int    DISPLAYSCREEN        ; /* 1=screen on,    0=screen off    */
extern int    LOCATIONLAB          ; /* 1=virtual world,0=actual vehicle */
extern int    BENCHTEST            ; /* 1=virtual world,0=actual vehicle */
extern int    TACTICAL              ; /* 1=tactical on,  0=tactical off   */
extern int    LOOPFOREVER          ; /* 1=repeat execution indefinitely */
extern int    LOOPFILEBACKUP       ; /* 1=backup files between replications*/

extern int    PARALLELPORTTRACE    ; /* 1=trace each char received at port */
extern int    SONARINSTALLED       ; /* 1=sonar head available for query */
extern int    SONARTRACE            ; /* 1=trace on,      0=trace off    */
extern double SONARHEADINGSTEP      ; /* degrees */
extern int    SONARSCANMODE         ; /* 1=forward, 0 = target tracking */
extern int    SONARSCANDIRECTION    ; /* 1=right, 2 = left */
extern double st1000_command        ; /* degrees commanded bearing */

extern int    ENTERCONTROLCONSTANTS ; /* 1 = manual entry, 0=default values */
extern int    SHOWCONTROLCONSTANTS  ; /* 1 = print constants, 0=default */
extern int    LOADCONTROLCONSTANTS  ; /* 1 = file entry, 0=default values */
extern int    REALTIME              ; /* 1 = real-time waits, 0 = no-pause */

extern int    DIVETRACKER           ; /* 1=no dive tracker means abort */
extern int    DEADRECKON            ; /* 1=dead reckon navigate, 0=regular */

```

```

extern int    DEADSTICKRUDDER      ; /* 1=use ordered rudder, 0 = control */
extern int    DEADSTICKPLANES      ; /* 1=use ordered planes, 0 = control */
extern int    INTEGRALDEPTHCONTROL ; /* 1=use PID, 0 = use PD */
extern double time_int_control_on  ; /* give PD a chance to get close */

/* untested */
extern int    SLIDINGMODECOURSE    ; /* 1=use sliding mode, 0 = control */

extern int    THRUSTERCONTROL      ; /* 1=use thrusters, 0=use propellers */
extern int    ROTATECONTROL        ; /* 1=use thrusters to rotate in place */
extern int    LATERALCONTROL       ; /* 1=use thrusters for lateral motion */
extern int    FOLLOWWAYPOINTMODE    ; /* 1= go to WAYPOINT without WAITs */
extern int    WAYPOINTCONTROL      ; /* 1= go to WAYPOINT */
extern int    HOVERCONTROL         ; /* 1=hover at WAYPOINT */

extern int    TARGETCONTROL        ; /* 1=hover relative to a target */
extern int    NEWTARGET            ; /* 1=target is new, 0=target is old */
extern int    TARGETPOINTING       ; /* 1=point at target if TARGETCONTROL */
extern int    TARGETEDGETRACK      ; /* 1=tracking on target edge only */
extern int    NEWTARGETSTATION     ; /* 1=new station keeping point */
extern int    RECOVERYCONTROL      ; /* 1=recovery in progress */
extern int    NEWRECOVERYCOMMAND   ; /* 1=command new, 0=in progress */

extern double SCANWIDTH            ; /* degrees full scan, centered on bow */

extern int    LEAK                 ; /* 1=water leak in progress */
extern int    HALTSCRIPT           ; /* 1=automatic shutdown criteria met */

extern int    DEATH_SPIRAL_RESET   ; /* 1=reset death spiral checker */

#ifdef os9
extern int    EMAIL                ; /* 1=send e-mail, 0=don't send e-mail */
#else
extern int    EMAIL                ; /* can't send email via OS-9 directly */
#endif

extern int    EMAIL_ENTERED        ; /* flag for first time through */

extern int    NOT_YET_REIMPLEMENTED ; /* code in block needs reverification */

extern int    ARCHAIC_IGNORE       ; /* code in block not valid, commented */

extern double TIMESTEP             ; /* time of a single closed loop */
                                   ; /* add code to warn if exceeded <<<< */
                                   ; /* units are seconds */

extern int    TACTICALPARSE        ; /* 1=tactical level parsing commands */
extern int    KEYBOARDINPUT        ; /* 1=read keyboard vice mission file */
extern int    HELPFILELAUNCHED     ; /* 1=mission.script.HELP already shown */
extern int    GPSFIXINPROGRESS     ; /* 1=wait GPS-FIX & restore z_command */
extern int    REPORTSTABLE         ; /* 1=tell when stable hover/waypt/gps */

extern int    REPLAY               ; /* 1=replay of existing telemetry file */
extern int    NOSCRIPT             ; /* 1=replay of existing telemetry file */
                                   ; /* with no accompanying script file */

/
*****
/* files and paths */

extern FILE * auvscriptfile;
extern FILE * auvordersfile;
extern FILE * auvdatafile;
extern FILE * auvtextfile;
extern FILE * telemetry_file;
extern FILE * controlconstantsinputfile;
extern FILE * controlconstantsoutputfile;

```

```

extern FILE * emailaddressfile;
extern FILE * st1000datafile;
extern FILE * targetdatafile;

/* FILE * serialtestfile; */

/* located here due to gespac cross-compiler */
extern char TELEMETRYFILENAME [255];

extern int    serialpath ;
extern int    sonarpath  ;

/
*****
/* Variables and data structures */
/* buffers of full strings for byte transfer to tactical level & disk file */
/* 'buffer' usually < 256, intentionally oversized in case of overflow error*/

extern time_t      system_time      ;
extern struct tm   *system_tmp      ;

/* partial structure template for the MFI (only interested in PIA for now)*/
struct MFI_PIA
{
    unsigned short pra;          /* port    register A - data direction A */
    unsigned short cra;          /* control register A */
    unsigned short prb;          /* port    register B - data direction B */
    unsigned short crb;          /* control register B */
};

/* dac: digital-analog converter */
/* adc: analog-digital converter */

/* 4 Channels of DAC ADA-1 DAC */
extern unsigned char *dac1_a ;

/* 8 Channels of DAC DAC-2B */
extern unsigned char *dac2b_a ;

/* 16 Channels of ADC ADA-1 */
extern unsigned char *adc1_a ;

/* 16 Channels of ADC ADC-2 */
extern unsigned short *adc2_a ;

extern unsigned char *via0 ;
extern unsigned char *via1 ;

extern unsigned char via0a_reg, via0b_reg;

extern int    telemetry_records_saved ;
extern int    mission_leg_counter    ;
extern int    replication_count       ;
extern int    end_test                ;
extern int    wrap_count              ;

extern double dt                      ; /* units are seconds */
extern double rpm                     ; /* -700..700 */

extern double dt_time                  ; /* dive tracker time */

extern double computer_voltage         ;
extern double motor_voltage            ;

```

```

extern double      vertical_thruster_volts ;/*intermediate calculation */
extern double      lateral_thruster_volts ; /* intermediate calculation */

extern double      main_motor_delta1      ;
extern double      main_motor_delta2      ;
extern int         main_motor_volt1       ;
extern int         main_motor_volt2       ;

extern unsigned short psi_bit_old          ;

extern double      dg_offset               ;

extern double      start_psi              ;

/* Used to estimate the X and Y position of the AUV */
extern double      X_est                  ;
extern double      Y_est                  ;
extern double      X_dot_est              ;
extern double      Y_dot_est              ;
extern double      u_est                  ;
extern double      v_est                  ;

/* control coefficients are based on standard units (degrees/feet/seconds) */
extern double      k_psi                  ;
extern double      k_r                    ;
extern double      k_v                    ;

extern double      k_z                    ;
extern double      k_w                    ;
extern double      k_theta                ;
extern double      k_q                    ;

extern double      k_thruster_psi         ;
extern double      k_thruster_r           ;
extern double      k_thruster_rotate      ;
extern double      k_thruster_lateral     ;
extern double      k_thruster_z           ;
extern double      k_thruster_w           ;
extern double      k_thruster_theta       ;

extern double      k_propeller_hover       ;
extern double      k_surge_hover           ;
extern double      k_propeller_current     ;

extern double      k_thruster_hover        ;
extern double      k_sway_hover            ;
extern double      k_thruster_current      ;

extern double      k_sigma_r              ;
extern double      k_sigma_psi            ;
extern double      eta_steering            ;
extern double      sigma                  ;

extern int         mission_legs_total      ;

/*      values initialized in parse_mission_script_commands () */
extern double      time_next_command       ; /* units are seconds */
extern double      time_gps_complete       ; /* units are seconds */
extern double      time_postgps_dive       ; /* units are seconds */
extern double      psi_command             ; /* degrees */
extern double      psi_command_hover       ; /* degrees */
extern double      theta_command           ; /* degrees */
extern double      x_command               ; /* feet */
extern double      y_command               ; /* feet */
extern double      z_command               ; /* feet */
extern double      stbd_rpm_command        ; /* -700..700 */
extern double      port_rpm_command        ; /* -700..700 */
extern double      planes_command          ; /* degrees */

```

```

extern double      rudder_command      ; /* degrees      */
extern double      rotate_command       ; /* degrees/sec   */
extern double      lateral_command      ; /* ft/sec       */

extern double      bow_lateral_thruster_command ; /* volts -24..24 */
extern double      stern_lateral_thruster_command ; /* volts -24..24 */
extern double      bow_vertical_thruster_command ; /* volts -24..24 */
extern double      stern_vertical_thruster_command ; /* volts -24..24 */

extern double      previous_z_command    ; /* feet          */
extern double      range_from_recovery_pt ; /* feet          */

extern double      gyro_error            ; /* degrees       */

extern double      waypoint_distance     ; /* feet          */
extern double      waypoint_angle        ; /* degrees       */

extern double      track_angle            ; /* degrees       */
extern double      along_track_distance  ; /* feet          */
extern double      cross_track_distance  ; /* feet          */
extern double      standoff_distance      ; /* feet          */

extern double      death_spiral_radius   ; /* feet          */

extern double      depth_error            ; /* feet          */
extern double      depth_cell_bias        ; /* feet          */
extern double      psi_error             ; /* degrees       */

/* parameters required for targetcontrol */
extern double      target_x              ; /* feet, world coords */
extern double      target_y              ; /* feet, world coords */
extern double      target_z              ; /* feet, world coords */
extern double      target_bearing        ; /* degrees        */
extern double      target_bearing_command ; /* degrees        */
extern double      target_bearing_dot    ; /* degrees per second */
extern double      target_range          ; /* feet           */
extern double      target_range_command  ; /* feet           */
extern double      target_range_dot      ; /* feet per second */

/* psi i-1 for differentiation of r needed because of busted gyro */
extern double      psi_im1               ; /* degrees        */

/* values used by kahlman depth filter */
extern int         kal_init_z            ;
extern double      thres_z               ;
extern double      z_kal                 ;
extern double      z_dot_kal             ;
extern double      z_ddot_kal            ;

extern int         roll_rate_0           ;
extern int         pitch_rate_0          ;
extern int         yaw_rate_0            ;
extern int         roll_0                ;
extern int         pitch_0               ;
extern int         z_val0                ;
extern int         sw1                   ;
extern int         error                  ;
extern int         range                  ;
extern int         bad_rng                ;
extern int         bad_updates            ;
extern int         range_index            ;
extern double      range1                 ;
extern double      range2                 ;
extern double      error1                  ;
extern double      error2                  ;
extern double      avg_rng                 ;
extern int         k_range                 ;
extern int         range_array [3000];

```

```

extern int          speed_array [11]          ;

extern int          PortAFlag                  ;

extern int          tick                      ;
extern int          curr_tick                  ;
extern int          tick1                     ;
extern int          tick2                     ;
extern int          i                         ;

extern int          mask                      ;
extern long         davedate                  ;
extern long         davetime                  ;
extern double       value                     ;
extern short        day                       ;
extern char         answer                    ;
extern int          start_dwell               ;

extern int          socket_descriptor         ;
extern int          socket_accepted           ;
extern int          socket_stream             ;

extern int          tactical_socket_descriptor;
extern int          tactical_socket_accepted ;
extern int          tactical_socket_stream    ;

extern int          socket_length              ; /* max allowed packet size*/

extern int          bytes_received             ;
extern int          bytes_read                 ;
extern int          bytes_written              ;
extern int          bytes_left                 ;
extern int          bytes_sent                 ;

/* char          buffer_array [FILEBUFFERSIZE][256]; -- not implemented */

extern char         buffer                    [MAXBUFFERSIZE + 10];
extern char         local_buffer              [MAXBUFFERSIZE + 10];
extern int          buffer_size                ;
extern int          buffer_max                 ;
extern int          buffer_index               ;
extern int          variables_parsed           ;

extern char         buffer_received            [MAXBUFFERSIZE + 10],
virtual_world_remote_host_name [60],
tactical_remote_host_name [60],
command_buffer      [MAXBUFFERSIZE + 10];

extern FILE         * netstat_fileptr;

extern struct sockaddr_in server_address;

extern struct hostent *server_entity;

extern char         email_address [81];

extern int          shutdown_signal_received ;
extern int          virtual_world_socket_opened ;
extern int          tactical_socket_opened ;

extern char         * ptr_index;

extern int          print_help;

extern double       speed_per_rpm; /* steady state: 2.0 feet/sec per 700 rpm */
/* -700..700 */

```

```

extern clock_t nextloopclock;
extern clock_t currentloopclock;

extern int     audible_command    ;

extern int     auvscriptfilequit  ;

extern double  AUV_oceancurrent_x; /* Ocean current rate along North-axis */
extern double  AUV_oceancurrent_y; /* Ocean current rate along East-axis */
extern double  AUV_oceancurrent_z; /* Ocean current rate along Depth-axis */

extern double  DiveTracker1_x;    /* DiveTracker1 transducer x (feet) */
extern double  DiveTracker1_y;    /* DiveTracker1 transducer y (feet) */
extern double  DiveTracker1_z;    /* DiveTracker1 transducer z (feet) */

extern double  DiveTracker2_x;    /* DiveTracker2 transducer x (feet) */
extern double  DiveTracker2_y;    /* DiveTracker2 transducer y (feet) */
extern double  DiveTracker2_z;    /* DiveTracker2 transducer z (feet) */

/*****

/* Dave's cats and dogs */

extern unsigned char *tim_lac1;
extern unsigned char *tim_lac2;
extern unsigned char *tim_lac3;

extern unsigned char tim_la_data_reg    ;
extern unsigned char tim_la_control_reg ;
extern unsigned char tim_la_aux_gates_reg ;

/* Dive Tracker Process Stuff */

#ifdef os9
extern int os9forkc();
extern char **environ;

extern char *dt_fork_parmptr;
extern char *argblk[];
extern int dt_pid;
extern int ul_pid;

#endif
/*****
#endif
/* GLOBALS_H */

```

```

/*****
/*
Program:      globals.c

Authors:      Don Brutzman, Duane Davis

Revised:      2 Agust 96

System:       AUV Gespac 68020/68030, OS-9 version 2.4
Compiler:     Gespac cc Kernighan & Richie (K&R) C

Compilation:  ftp>      put globals.c
               auvsim1> chd execution
               auvsim1> make -k2f execution
               auvsim1> make      execution

               [Irix ]   fletch> make execution

Purpose:      Allows repeated use of global variables global.c via global.h
               in order to prevent compiler warnings

               See statevector.c/statevector.h for other global variables

*/

/*****
#ifndef GLOBALS_C
#define GLOBALS_C
/*****

#include "globals.h"

/*****
/* Program configuration flags */

int  TRACE                = 0;  /* 1=trace on,      0=trace off      */
int  DISPLAYSCREEN        = 1;  /* 1=screen on,    0=screen off    */
int  LOCATIONLAB          = 1;  /* 1=virtual world,0=actual vehicle */
int  BENCHTEST            = 0;  /* 1=virtual world,0=actual vehicle */
int  TACTICAL             = 0;  /* 1=tactical on,  0=tactical off   */
int  LOOPFOREVER          = 0;  /* 1=repeat execution indefinitely */
int  LOOPFILEBACKUP       = 1;  /* 1=backup files between replications*/

int  PARALLELPORTRTRACE   = 0;  /* 1=trace each char received at port */
int  SONARINSTALLED       = 1;  /* 1=sonar head available for query   */
int  SONARTRACE           = 0;  /* 1=trace on,      0=trace off       */
int  SONARSCANMODE        = 1;  /* 1=forward, 2=target tracking       */
double SONARHEADINGSTEP= 0.9; /* degrees */
int  SONARSCANDIRECTION   = 1;  /* 1=right, 2 = left                  */
double st1000_command     = 0.0; /* degrees commanded bearing         */

int  ENTERCONTROLCONSTANTS = 0;  /* 1 = manual entry, 0=default values*/
int  SHOWCONTROLCONSTANTS  = 0;  /* 1 = print constants, 0=default     */
int  LOADCONTROLCONSTANTS  = 1;  /* 1 = file entry, 0=default values   */
int  REALTIME              = 0;  /* 1 = real-time waits, 0 = no-pause  */

int  DIVETRACKER           = 0;  /* 1=dive tracker being used         */

int  DEADRECKON            = 0;  /* 1=dead reckon navigate, 0=regular  */
int  DEADSTICKRUDDER       = 0;  /* 1=use ordered rudder, 0 = control  */
int  DEADSTICKPLANES       = 0;  /* 1=use ordered planes, 0 = control  */
int  INTEGRALDEPTHCONTROL  = 0;  /* 1=use PID, 0 = use PD              */
double time_int_control_on = 1000000.0; /* give PD a chance to get close*/

/* untested */
int  SLIDINGMODECOURSE     = 0;  /* 1=use sliding mode, 0 = control    */

```



```

int    THRUSTERCONTROL      = 0; /* 1=use thrusters, 0=use propellers */
int    ROTATECONTROL        = 0; /* 1=use thrusters to rotate in place */
int    LATERALCONTROL       = 0; /* 1=use thrusters for lateral motion */
int    FOLLOWWAYPOINTMODE   = 0; /* 1= go to WAYPOINT without WAITs */
int    WAYPOINTCONTROL      = 0; /* 1= go to WAYPOINT */
int    HOVERCONTROL         = 0; /* 1=hover at WAYPOINT */

int    TARGETCONTROL        = 0; /* 1=hover relative to a target */
int    NEWTARGET            = 0; /* 1=target is new, 0=target is old */
int    TARGETPOINTING       = 0; /* 1=point at target if TARGETCONTROL */
int    TARGETEDGETRACK      = 0; /* 1=tracking on target edge only */
int    NEWTARGETSTATION     = 0; /* 1=new station keeping point */
int    RECOVERYCONTROL      = 0; /* 1=recovery in progress */
int    NEWRECOVERYCOMMAND   = 1; /* 1=command new, 0=in progress */

double SCANWIDTH            = 30.0; /* degrees full scan, centered on bow */
                                   /* 90 degrees takes ~15 seconds */
                                   /* 45 degrees probably optimal */

int    LEAK                 = 0; /* 1=water leak in progress */
int    HALTSCRIPT           = 0; /* 1=automatic shutdown criteria met */

int    DEATH_SPIRAL_RESET   = 1; /* 1=reset death spiral checker */

#ifndef os9
int    EMAIL                = 0; /* 1=send e-mail, 0=don't send e-mail */
#else
int    EMAIL                = 0; /* can't send email via OS-9 directly */
#endif

int    EMAIL_ENTERED        = 0; /* flag for first time through */

int    NOT_YET_REIMPLEMENTED = 0; /* code in block needs reverification */
int    ARCHAIC_IGNORE       = 0; /* code in block not valid, commented */

double TIMESTEP             = 0.15; /* time of a single closed loop */
                                   /* add code to warn if exceeded <<<< */
                                   /* units are seconds */

int    TACTICALPARSE        = 0; /* 1=tactical level parsing commands */
int    KEYBOARDINPUT        = 0; /* 1=read keyboard vice mission file */
int    HELPPFILELAUNCHED   = 0; /* 1=mission.script.HELP already
shown*/
int    GPSFIXINPROGRESS     = 0; /* 1=wait GPS-FIX & restore z_command */
int    REPORTSTABLE        = 0; /* 1=tell when stable hover/waypt/gps */

int    REPLAY               = 0; /* 1=replay of existing telemetry file */
int    NOSCRIPT             = 0; /* 1=replay of existing telemetry file */
                                   /* with no accompanying script file */

/*****
/* files and paths
*/

FILE * auvscriptfile;
FILE * auvordersfile;
FILE * auvdatafile;
FILE * auvtextfile;
FILE * telemetry_file;
FILE * controlconstantsinputfile;
FILE * controlconstantsoutputfile;
FILE * emailaddressfile;
FILE * st1000datafile;
FILE * targetdatafile;

/* FILE * serialtestfile; */

char TELEMETRYFILENAME [255]; /* located here due to gespac cross-compiler */

```

```

int serialpath = 0;
int sonarpath = 0;

/*****
/* Variables and data structures */
/* buffers of full strings for byte transfer to tactical level & disk file */
/* 'buffer' usually < 256, intentionally oversized in case of overflow error*/

time_t      system_time      = 0;
struct tm   *system_tmp      = 0;

/* dac: digital-analog converter */
/* adc: analog-digital converter */

/* 4 Channels of DAC ADA-1 DAC -- updated */
unsigned char *dac1_a = (unsigned char *) DAC1_ADDR;

/* 8 Channels of DAC DAC-2B -- updated */
unsigned char *dac2b_a = (unsigned char *) DAC2B_ADDR;

/* 16 Channels of ADC ADA-1 -- updated */
unsigned char *adc1_a = (unsigned char *) ADC1_ADDR;

/* 16 Channels of ADC ADC-2 -- updated */
unsigned short *adc2_a = (unsigned short *) ADC2_ADDR;

unsigned char *via0 = (unsigned char *) VIA0_ADDR;
unsigned char *via1 = (unsigned char *) VIA1_ADDR;

unsigned char via0a_reg, via0b_reg;

int      telemetry_records_saved = 0;
int      mission_leg_counter     = 0;
int      replication_count       = 1;
int      end_test                = FALSE;
int      wrap_count              = 0;

double   dt                      = 0.15; /* units are seconds */
double   rpm                    = 0.0; /* +-700 rpm == +-2 ft/sec */
/* (steady state) */

double   dt_time                ; /* dive tracker time */

double   computer_voltage       = 24.0;
double   motor_voltage          = 24.0;

double   vertical_thruster_volts = 0.0; /*intermediate calculation */
double   lateral_thruster_volts = 0.0; /*intermediate calculation */

double   main_motor_delta1      = 0.0;
double   main_motor_delta2      = 0.0;
int      main_motor_volt1       = 512;
int      main_motor_volt2       = 512;

unsigned short psi_bit_old      = 0;

double   dg_offset              = 0.0;

double   start_psi              = 0.0; /*initial heading in degrees */

/* Used to estimate the X and Y position of the AUV */
double   X_est                  = 0.0;
double   Y_est                  = 0.0;
double   X_dot_est              = 0.0;
double   Y_dot_est              = 0.0;

```

```

double      u_est      = 0.0;
double      v_est      = 0.0;

/*control coefficients are based on standard units (degrees/feet/seconds)*/
double      k_psi      = 0.0;
double      k_r         = 0.0;
double      k_v         = 0.0;

double      k_z         = 0.0;
double      k_w         = 0.0;
double      k_theta     = 0.0;
double      k_q         = 0.0;

double      k_thruster_psi = 0.0;
double      k_thruster_r  = 0.0;
double      k_thruster_rotate = 0.0;
double      k_thruster_lateral = 0.0;
double      k_thruster_z  = 0.0;
double      k_thruster_w  = 0.0;
double      k_thruster_theta = 0.0;

double      k_propeller_hover = 0.0;
double      k_surge_hover   = 0.0;
double      k_propeller_current = 0.0;

double      k_thruster_hover = 0.0;
double      k_sway_hover     = 0.0;
double      k_thruster_current = 0.0;

double      k_sigma_r      = 12.0;
double      k_sigma_psi    = 28.87;
double      eta_steering   = 0.1;
double      sigma          = 0.0;

int         mission_legs_total = 0;

/* values initialized in parse_mission_script_commands () */
double      time_next_command = 0.0; /* units are seconds */
double      time_gps_complete = 0.0; /* units are seconds */
double      time_postgps_dive = 0.0; /* units are seconds */
double      psi_command       = 0.0; /* degrees */
double      psi_command_hover = 0.0; /* degrees */
double      theta_command     = 0.0; /* degrees */
double      x_command         = 0.0; /* feet */
double      y_command         = 0.0; /* feet */
double      z_command         = 0.0; /* feet */
double      stbd_rpm_command  = 0.0; /* -700..700 */
double      port_rpm_command  = 0.0; /* -700..700 */
double      planes_command    = 0.0; /* degrees */
double      rudder_command    = 0.0; /* degrees */
double      rotate_command    = 0.0; /* degrees/sec */
double      lateral_command   = 0.0; /* ft/sec */

double      bow_lateral_thruster_command = 0.0; /* volts -24..24 */
double      stern_lateral_thruster_command = 0.0; /* volts -24..24 */
double      bow_vertical_thruster_command = 0.0; /* volts -24..24 */
double      stern_vertical_thruster_command = 0.0; /* volts -24..24 */

double      previous_z_command = 0.0; /* feet */
double      range_from_recovery_pt = 0.0; /* feet */

double      gyro_error        = 0.0; /* degrees */

double      waypoint_distance = 0.0; /* feet */
double      waypoint_angle    = 0.0; /* degrees */

double      track_angle       = 0.0; /* degrees */
double      along_track_distance = 0.0; /* feet */

```

```

double      cross_track_distance    = 0.0; /* feet      */
double      standoff_distance       = 2.5; /* feet      */

double      death_spiral_radius     = 15.0; /* feet      */

double      depth_error              ; /* feet      */
double      depth_cell_bias          = 0.0; /* feet      */
double      psi_error                = 0.0; /* degrees   */

/* parameters required for targetcontrol */
double      target_x                 = 0.0; /* feet, world coords */
double      target_y                 = 0.0; /* feet, world coords */
double      target_z                 = 0.0; /* feet, world coords */
double      target_bearing           = 0.0; /* degrees    */
double      target_bearing_command   = 0.0; /* degrees    */
double      target_bearing_dot       = 0.0; /* degrees per second */
double      target_range             = 0.0; /* feet      */
double      target_range_command     = 0.0; /* feet      */
double      target_range_dot         = 0.0; /* feet per second */

/* psi i-1 for differentiation of r needed because of busted gyro */
double      psi_im1                  = 0.0; /* degrees    */

/* values used by kahlman depth filter */
int         kal_init_z               = TRUE;
double      thres_z                  = 1.0;
double      z_kal                    = 0.0;
double      z_dot_kal                = 0.0;
double      z_ddot_kal               = 0.0;

int         roll_rate_0              = 0;
int         pitch_rate_0             = 0;
int         yaw_rate_0               = 0;
int         roll_0                   = 0;
int         pitch_0                  = 0;
int         z_val0                   = 0;
int         sw1                      = 0;
int         error                    = 0;
int         range                    = 0;
int         bad_rng                  = 0;
int         bad_updates              = 0;
int         range_index              = 0;
double      range1                   = 0.0;
double      range2                   = 0.0;
double      error1                   = 0.0;
double      error2                   = 0.0;
double      avg_rng                  = 0.0;
int         k_range                  = 0;
int         range_array [3000];

int         speed_array [11];

int         PortAFlag                = 0;

int         tick                    = 0;
int         curr_tick                = 0;
int         tick1                    = 0;
int         tick2                    = 0;
int         i                       = 0;

int         mask                    = 0x0000ffff;
long        davedate                 = 0;
long        davetime                 = 0;
double      value                    = 0.0;
short       day                     = 0;
char        answer                   = ' ';
int         start_dwell              = 0;

```

```

int          socket_descriptor      = 0;
int          socket_accepted       = 0;
int          socket_stream         = 0;

int          tactical_socket_descriptor = 0;
int          tactical_socket_accepted = 0;
int          tactical_socket_stream  = 0;

int          socket_length         = MAXBUFFERSIZE;
int          buffer_max            = MAXBUFFERSIZE;

/* char      buffer_array [FILEBUFFERSIZE][256]; -- not implemented */

char         buffer                 [MAXBUFFERSIZE + 10];
char         local_buffer           [MAXBUFFERSIZE + 10];

int          buffer_size            = 0;
int          buffer_index           = 0;
int          variables_parsed       = 0;

char         buffer_received        [MAXBUFFERSIZE + 10],
virtual_world_remote_host_name [60],
          tactical_remote_host_name [60],
command_buffer [MAXBUFFERSIZE + 10];

int          bytes_received         = 0;
int          bytes_read              = 0;
int          bytes_written           = 0;
int          bytes_left              = 0;
int          bytes_sent              = 0;

FILE         * netstat_fileptr;

struct sockaddr_in  server_address;

struct hostent *server_entity;

char          email_address [81];

int           shutdown_signal_received = FALSE;
int           virtual_world_socket_opened = FALSE;
int           tactical_socket_opened = FALSE;

char * ptr_index;

int          print_help              = FALSE;

double       speed_per_rpm = 2.0 / 700.0 ; /* steady state:
                                           2.0 feet/sec per 700 rpm */
                                           /* -700..700 */

clock_t      nextloopclock           = 0;
clock_t      currentloopclock        = 0;

int          audible_command         = TRUE;

int          auvscriptfilequit       = FALSE;

double AUV_oceancurrent_x = 0.0; /* Ocean current rate along North-axis */
double AUV_oceancurrent_y = 0.0; /* Ocean current rate along East-axis */
double AUV_oceancurrent_z = 0.0; /* Ocean current rate along Depth-axis */

double DiveTracker1_x; /* DiveTracker1 transducer x (feet) */
double DiveTracker1_y; /* DiveTracker1 transducer y (feet) */
double DiveTracker1_z; /* DiveTracker1 transducer z (feet) */

double DiveTracker2_x; /* DiveTracker2 transducer x (feet) */
double DiveTracker2_y; /* DiveTracker2 transducer y (feet) */

```

```

double DiveTracker2_z;          /* DiveTracker2 transducer z (feet) */
/*****
/* Dave's cats and dogs */

unsigned char *tim_1ac1 = (unsigned char *) TIM_1AC_1;
unsigned char *tim_1ac2 = (unsigned char *) TIM_1AC_2;
unsigned char *tim_1ac3 = (unsigned char *) TIM_1AC_3;

unsigned char tim_1a_data_reg    = TIM_1AC_DATA_REG;
unsigned char tim_1a_control_reg = TIM_1AC_CONTROL_REG;
unsigned char tim_1a_aux_gates_reg = TIM_1AC_AUX_GATES_REG;

#ifdef os9

char *dt_fork_parmptr;
char *argblk[] = {
    "unlink",
    "DT2CL",
    0,
};
int dt_pid;
int ul_pid;

#endif
/*****
#endif
/* GLOBALS_C */

```

```

/*****
/*
Program:      external_functions.c

Authors:      Don Brutzman

Revised:      2 August 96

System:       AUV Gespac 68020/68030, OS-9 version 2.4
Compiler:     Gespac cc Kernighan & Richie (K&R) C

Compilation:  ftp>      put external_functions.c
               auvsim1> chd execution
               [68020]   auvsim1> make -k2f execution
               [68030]   auvsim1> make      execution

               [Irix ]   fletch> make execution

Purpose:      Reduce size of execution.c to allow OS-9 C compiler to work

               Make functions globally available for tactical level
               in order to guarantee compatibility

Note that %F double formats are used instead of %lf on scanf() and sscanf()
calls for OS-9 compatibility.  SGI C compiler does not complain.
gcc on Sun does fail at run-time, so ifdef's are used to support
the proper format string.

*/

/*****
/* external_functions.c */

#include "globals.h"
#include "statevector.h"
#include "defines.h"

/*****
/* OS-9 ~ specific function compatibility */
/* (mostly stubs to permit compilation under Unix) */

#ifdef os9

void    tsleep    (unsigned svalue) { /* null body */}

void    _sysdate  (format, time, date, day, tick)
           int format, *time, *date, *tick; short *day;
           { /* null body */}

double  pow       (xx, yy)
           double xx, yy;
           {return exp (yy * log (xx));}

int     _gs_rdy   (path) int path; { return 0; } /* bytes waiting on path */

int     _gs_opt   (path, buffer)
           int path; struct sgbuf *buffer;
           { return 0; }

int     _ss_opt   (path, buffer)
           int path; struct sgbuf *buffer;
           { return 0; }

#endif

/*****

double    degrees

           ();

```

```

double      radians      ();
double      normalize    ();
double      normalize2   ();
double      radian_normalize ();
double      radian_normalize2 ();
void        clamp        ();

double      atan2z       ();

#ifdef os9
double      atan2        ();
double      sinh         ();
double      cosh         ();
double      tanh         ();
#endif

void        build_telemetry_string    ();
void        parse_telemetry_string    ();
void        read_telemetry_string     ();

void        open_tactical_socket      ();
void        shutdown_tactical_socket ();
void        send_buffer_to_tactical_socket ();
void        get_string_from_tactical_socket ();

void        record_data_on            ();
void        record_data_off          ();

void        cage_dg                  ();
void        uncage_dg                 ();

int         detect_death_spiral      ();

double      dsign                   ();
double      dtanh                   ();

/*****/
double degrees (rads)      /* radians input */
{
    double rads;
    {
        return rads * 180.0 / PI;
    }
}
/*****/

double radians (degs)      /* degrees input*/
{
    double degs;
    {
        return degs * PI / 180.0;
    }
}
/*****/

double normalize (degs)      /* degrees input*/
{
    double degs;
    {
        double result = degs;

        while (result < 0.0) result += 360.0;
        while (result >= 360.0) result -= 360.0;

        return result;
    }
}
/*****/

double normalize2 (degs)      /* degrees input*/
{
    double degs;
    {

```



```

    double result = degs;

    while (result <= -180.0) result += 360.0;
    while (result > 180.0) result -= 360.0;

    return result;
}
/*****

double radian_normalize (rads)      /* radians input*/
    double rads;
{
    double result = rads;

    while (result < 0.0) result += 2.0 * PI;
    while (result >= 2.0 * PI) result -= 2.0 * PI;

    return result;
}
/*****

double radian_normalize2 (rads)      /* radians input*/
    double rads;
{
    double result = rads;

    while (result <= -PI) result += 2.0 * PI;
    while (result > PI) result -= 2.0 * PI;

    return result;
}
/*****

void clamp (clampee, absolute_min, absolute_max, name)
    double * clampee;
    double absolute_min;
    double absolute_max;
    char * name;
{
    double new_value, local_min, local_max;

    if ((absolute_max == 0.0) && (absolute_min == 0.0)) return; /* no clamp */
    if (absolute_max >= absolute_min) /*ensure min & max used in proper order */
    {
        local_min = absolute_min;
        local_max = absolute_max;
    }
    else
    {
        local_min = absolute_max;
        local_max = absolute_min;
    }
    if ((* clampee) > local_max)
    {
        new_value = local_max;

        if (TRACE && DISPLAYSCREEN)
            printf ("[clamping %s from %5.3f to %5.3f]\n",
                    name, * clampee, new_value);

        * clampee = new_value;
    }
    if ((* clampee) < local_min)
    {
        new_value = local_min;

        if (TRACE && DISPLAYSCREEN)

```

```

        printf ("[clamping %s from %5.3f to %5.3f]\n",
                name, * clampee, new_value);

        * clampee = new_value;
    }
}
/*****

double atan2z (x, y)
    double y; double x;

{ /* more reliable (0,0)-protected atan2 due to cross-platform noncompliance
*/

    if ((fabs (x) == 0.0) && (fabs (y) == 0.0)) return 0.0;
    else return atan2 (x, y);
}
/*****
#ifdef os9

/* thanks to Michael Olberg    Oct 20, 94    olberg@bele.oso.chalmers.se    */
/* modified to map (0,0) to 0.0    */

double atan2 (y, x)
    double y; double x;
{
    if (TRACE && DISPLAYSCREEN)
        printf ("[atan2 (%5.3f, %5.3f)]\n", y, x);

    if (fabs (x) == 0.0) {
        if (y < 0.0) return(-PI/2.0);
        else if (fabs (y) == 0.0) return 0.0;
        else return( PI/2.0);
    } else {
        if (x < 0.0) {
            if (y < 0.0) return(atan(y/x)-PI);
            else return(atan(y/x)+PI);
        } else return(atan(y/x));
    }
}

/* as to the tanh you will simply have to use */

double sinh (x)
    double x;
{
    return (exp(x) - exp(-x))/2.0;
}

double cosh (x)
    double x;
{
    return (exp(x) + exp(-x))/2.0;
}

double tanh (x)
    double x;
{
    return sinh(x)/cosh(x);
}

#endif

/*****

void build_telemetry_string (telemetry_buffer_ptr)
    char * telemetry_buffer_ptr;

```



```

r = (AUV_r_temp);
x_dot = (AUV_x_dot_temp);
y_dot = (AUV_y_dot_temp);
z_dot = (AUV_z_dot_temp);
phi_dot = (AUV_phi_dot_temp);
theta_dot = (AUV_theta_dot_temp);
psi_dot = (AUV_psi_dot_temp);
delta_rudder = (AUV_delta_rudder_temp);
delta_planes = (AUV_delta_planes_temp);
port_rpm = (AUV_port_rpm_temp);
stbd_rpm = (AUV_stbd_rpm_temp);
AUV_bow_vertical = (AUV_bow_vertical_temp);
AUV_stern_vertical = (AUV_stern_vertical_temp);
AUV_bow_lateral = (AUV_bow_lateral_temp);
AUV_stern_lateral = (AUV_stern_lateral_temp);
AUV_ST1000_bearing = normalize(AUV_ST1000_bearing_temp);
AUV_ST1000_range = (AUV_ST1000_range_temp);
AUV_ST1000_strength = (AUV_ST1000_strength_temp);
AUV_ST725_bearing = normalize(AUV_ST725_bearing_temp);
AUV_ST725_range = (AUV_ST725_range_temp);
AUV_ST725_strength = (AUV_ST725_strength_temp);
divetracker_range1 = (AUV_divetracker_range1_temp);
divetracker_range2 = (AUV_divetracker_range2_temp);
}
else if ((variables_parsed != STATEVECTORSIZE) && (variables_parsed != -1))
{
    strcpy (command_buffer, passed_buffer_ptr);
    parse_mission_script_commands ();

    /* if (DISPLAYSCREEN) printf ("\nGarble problem in
buffer_received !!! variables parsed = %d\n%s\n",
variables_parsed, passed_buffer_ptr); TRACE = TRUE; */ }

if (TRACE && DISPLAYSCREEN)
{
    printf ("\nfrom telemetry buffer state variables:");
    printf ("\n %s t=%5.3f x=%5.3f y=%5.3f z=%5.3f ",
keyword, t, y, z);
    printf ("phi=%5.3f theta=%5.3f psi=%5.3f ",
phi, theta, psi);
    printf ("paddlewheel speed=%5.3f ",
speed);
    printf ("u=%5.3f v=%5.3f w=%5.3f p=%5.3f q=%5.3f r=%5.3f ",
u, v, w, p, q, r);
    printf ("x_dot=%5.3f y_dot=%5.3f z_dot=%5.3f ",
x_dot, y_dot, z_dot);
    printf ("phi_dot=%5.3f theta_dot=%5.3f psi_dot=%5.3f ",
phi_dot, theta_dot, psi_dot);
    printf ("delta_rudder=%5.3f delta_planes=%5.3f ",
delta_rudder, delta_planes);
    printf ("port_rpm=%5.3f stbd_rpm=%5.3f ",
port_rpm, stbd_rpm);
    printf ("bow_vertical=%5.3f stern_vertical=%5.3f ",
AUV_bow_vertical, AUV_stern_vertical);
    printf ("bow_lateral=%5.3f stern_lateral=%5.3f ",
AUV_bow_lateral, AUV_stern_lateral);
    printf ("ST1000 b/r/s %5.3f %5.3f %5.3f, ST725 b/r/s %5.3f %5.3f %5.3f",
AUV_ST1000_bearing, AUV_ST1000_range, AUV_ST1000_strength,
AUV_ST725_bearing, AUV_ST725_range, AUV_ST725_strength);
    printf ("divetracker_range1=%5.3f divetracker_range2=%5.3f ",
divetracker_range1, divetracker_range2);
    printf ("", [current time %d %d %d] \n",
system_tmp->tm_hour, system_tmp->tm_min, system_tmp->tm_sec);
}

/* keep all telemetry variables in degrees */

```



```

        &AUV_x_dot_temp,          &AUV_y_dot_temp,    &AUV_z_dot_temp,
        &AUV_phi_dot_temp,        &AUV_theta_dot_temp, &AUV_psi_dot_temp,
        &AUV_delta_rudder_temp,  &AUV_delta_planes_temp,
        &AUV_port_rpm_temp,      &AUV_stbd_rpm_temp,
        &AUV_bow_vertical_temp,  &AUV_stern_vertical_temp,
        &AUV_bow_lateral_temp,  &AUV_stern_lateral_temp,
        &AUV_ST1000_bearing_temp, &AUV_ST1000_range_temp,
        &AUV_ST1000_strength_temp, &AUV_ST725_range_temp,
        &AUV_ST725_bearing_temp, &AUV_ST725_strength_temp,
        &AUV_divetracker_range1_temp, &AUV_divetracker_range2_temp);

if (variables_parsed == STATEVECTORSIZE) /* transfer OK, keep new values */
{
    t = (AUV_time_temp);
    x = (AUV_x_temp);
    y = (AUV_y_temp);
    z = (AUV_z_temp);
    phi = (AUV_phi_temp);
    theta = (AUV_theta_temp);
    psi = (AUV_psi_temp);
    speed = (AUV_speed_temp);
    u = (AUV_u_temp);
    v = (AUV_v_temp);
    w = (AUV_w_temp);
    p = (AUV_p_temp);
    q = (AUV_q_temp);
    r = (AUV_r_temp);
    x_dot = (AUV_x_dot_temp);
    y_dot = (AUV_y_dot_temp);
    z_dot = (AUV_z_dot_temp);
    phi_dot = (AUV_phi_dot_temp);
    theta_dot = (AUV_theta_dot_temp);
    psi_dot = (AUV_psi_dot_temp);
    delta_rudder = (AUV_delta_rudder_temp);
    delta_planes = (AUV_delta_planes_temp);
    port_rpm = (AUV_port_rpm_temp);
    stbd_rpm = (AUV_stbd_rpm_temp);
    AUV_bow_vertical = (AUV_bow_vertical_temp);
    AUV_stern_vertical = (AUV_stern_vertical_temp);
    AUV_bow_lateral = (AUV_bow_lateral_temp);
    AUV_stern_lateral = (AUV_stern_lateral_temp);
    AUV_ST1000_bearing = normalize(AUV_ST1000_bearing_temp);
    AUV_ST1000_range = (AUV_ST1000_range_temp);
    AUV_ST1000_strength = (AUV_ST1000_strength_temp);
    AUV_ST725_bearing = normalize(AUV_ST725_bearing_temp);
    AUV_ST725_range = (AUV_ST725_range_temp);
    AUV_ST725_strength = (AUV_ST725_strength_temp);
    divetracker_range1 = (AUV_divetracker_range1_temp);
    divetracker_range2 = (AUV_divetracker_range2_temp);
}
else end_test = TRUE;

if (TRACE && DISPLAYSCREEN)
{
    printf ("\nRead from telemetry file:");
    printf ("\n %s t=%5.3f x=%5.3f y=%5.3f z=%5.3f ",
        keyword, t, y, z);
    printf ("phi=%5.3f theta=%5.3f psi=%5.3f ",
        phi, theta, psi);
    printf ("paddlewheel speed=%5.3f ",
        speed);
    printf ("u=%5.3f v=%5.3f w=%5.3f p=%5.3f q=%5.3f r=%5.3f ",
        u, v, w, p, q, r);
    printf ("x_dot=%5.3f y_dot=%5.3f z_dot=%5.3f ",
        x_dot, y_dot, z_dot);
}

```

```

printf ("phi_dot=%5.3f theta_dot=%5.3f psi_dot=%5.3f ",
        phi_dot,      theta_dot,      psi_dot);
printf ("delta_rudder=%5.3f delta_planes=%5.3f ",
        delta_rudder, delta_planes);
printf ("port_rpm=%5.3f stbd_rpm=%5.3f ",
        port_rpm,     stbd_rpm);
printf ("bow_vertical=%5.3f stern_vertical=%5.3f ",
        AUV_bow_vertical, AUV_stern_vertical);
printf ("bow_lateral=%5.3f stern_lateral=%5.3f ",
        AUV_bow_lateral, AUV_stern_lateral);
printf ("ST1000 b/r/s %5.3f %5.3f %5.3f, ST725 b/r/s %5.3f %5.3f %5.3f",
        AUV_ST1000_bearing, AUV_ST1000_range, AUV_ST1000_strength,
        AUV_ST725_bearing, AUV_ST725_range, AUV_ST725_strength);
printf ("divetracker_range1=%5.3f divetracker_range2=%5.3f ",
        divetracker_range1, divetracker_range2);
printf (" [current time %d %d %d] \n",
        system_tmp->tm_hour, system_tmp->tm_min, system_tmp->tm_sec);
}

/* keep all telemetry variables in degrees */
phi      = normalize2 (phi );
theta    = normalize2 (theta);
psi      = normalize (psi );
phi_dot  = normalize2 (phi_dot);
theta_dot = normalize2 (theta_dot);
psi_dot  = normalize2 (psi_dot);
p        = normalize2 (p);
q        = normalize2 (q);
r        = normalize2 (r);
delta_rudder = normalize2 (delta_rudder);
delta_planes = normalize2 (delta_planes);

if (TRACE && DISPLAYSCREEN) printf ("[finish read_telemetry_string ()]\n");

return;
}
/*****

void open_tactical_socket ()      /* see os9sender.c for original code */
{
    if (TRACE && DISPLAYSCREEN)
        printf ("[start open_tactical_socket ()]\n");

    /* Initialize communications blocks */

    /* Initialize both client & server *****/

    /* Signal handlers for termination to override net_open () and net_close () */
    /* signal handlers. Otherwise you are unable to ^C kill this program. */

#ifdef os9
    signal (SIGHUP, shutdown_tactical_socket);/* hangup */
    signal (SIGINT, shutdown_tactical_socket);/* interrupt character */
    signal (SIGKILL, shutdown_tactical_socket);/* kill signal from Unix */
    signal (SIGPIPE, shutdown_tactical_socket);/* broken pipe from other host */
    signal (SIGTERM, shutdown_tactical_socket);/* software termination */
#endif

    /* Initialize sender *****/

    /* start by finding default/desired remote host to connect to */
    {
        server_entity = gethostbyname (tactical_remote_host_name);
        if (server_entity == NULL)
        {
            if (DISPLAYSCREEN)
            {

```



```

    {
        if (TRACE && DISPLAYSCREEN)
        {
            printf ("[shutdown_tactical_socket shutdown");
            printf (" (tactical_socket_stream, 2) failed]\n");
        }

        kill_return_value = kill (tactical_socket_stream, SIGKILL);

        if (TRACE && DISPLAYSCREEN)
        {
            printf ("[shutdown_tactical_socket kill (tactical_socket_stream,");
            printf (" SIGKILL) returned %d]\n", kill_return_value);
        }
    }
}

if (TRACE && DISPLAYSCREEN)
    printf ("[shutdown_tactical_socket return]\n");

return;
} /* end shutdown_tactical_socket () */

/*****

void send_buffer_to_tactical_socket () /* see os9sender.c for orig. code */
{
    if (HALTSCRIPT) return;

    bytes_left      = socket_length;
    bytes_written    = 0;
    ptr_index       = buffer; /* this global string is the data to be sent */

    if (tactical_socket_opened == FALSE)
    {
        if (TRACE && DISPLAYSCREEN)
        {
            printf ("[send_buffer_to_tactical_socket: ");
            printf ("tactical_socket_opened == FALSE, returning]\n");
        }
        return;
    }
    if (TRACE && DISPLAYSCREEN)
        printf ("[send_buffer_to_tactical_socket start ...]\n");

    while ((bytes_left > 0) && (bytes_written >= 0)) /* write loop *****/
    {
        bytes_sent = write (tactical_socket_stream, ptr_index, bytes_left);

        if (bytes_sent < 0) bytes_written = bytes_sent;
        else if (bytes_sent > 0)
        {
            bytes_left    -= bytes_sent;
            bytes_written += bytes_sent;
            ptr_index      += bytes_sent;
        }

        if (LOCATIONLAB && TRACE && DISPLAYSCREEN)
        {
            printf ("[record_data send_telemetry_to_server loop");
            printf (" bytes sent = %d]\n", bytes_sent);
        }
    }
    if (bytes_written < 0)
    {
        HALTSCRIPT = TRUE; /* loss of socket comms with tactical level */

        if (LOCATIONLAB && DISPLAYSCREEN)

```

```

        printf ("[record_data send_telemetry_to_server () send failed, ");
        printf ("%d bytes_written]\n", bytes_written);
    }
    /* error message needed on (open) output file <<<<<<<<<<<<<<<<<< */
}
else if (LOCATIONLAB && TRACE && DISPLAYSCREEN)
{
    printf ("[record_data send_telemetry_to_server total bytes sent");
    printf (" = %d]\n", bytes_written);
}

/* Check termination *****/
if (strncmp (buffer, "shutdown", 8) == 0)
{
    if (TRACE && DISPLAYSCREEN) printf
        ("[send_buffer_to_tactical_socket: shutdown_signal_received]\n");
    shutdown_tactical_socket ();
}
if (TRACE && DISPLAYSCREEN)
    printf ("[send_buffer_to_tactical_socket return]\n");

return;
} /* end send_buffer_to_tactical_socket () */

/*****/
void get_string_from_tactical_socket () /* see os9sender.c for original */
{
    if (tactical_socket_opened == FALSE)
    {
        if (TRACE && DISPLAYSCREEN)
            printf ("[get_string_from_tactical_socket not open, ignored]\n");
        return;
    }
    if (TRACE && DISPLAYSCREEN)
        printf ("[get_string_from_tactical_socket start ...]\n");

    /* listen to remote host, relay to local network/program */

    bytes_left      = socket_length;
    bytes_received   = 0;
    ptr_index       = command_buffer; /* command_buffer is where results go */

    while ((bytes_left > 0) && (bytes_received >= 0)) /* read loop *****/
    {
        bytes_read = read (tactical_socket_stream, ptr_index, bytes_left);

        if (bytes_read < 0) bytes_received = bytes_read;
        else if (bytes_read > 0)
        {
            bytes_left      -= bytes_read;
            bytes_received += bytes_read;
            ptr_index       += bytes_read;
        }
        if (TRACE && DISPLAYSCREEN)
        {
            printf ("[get_string_from_tactical_socket receiver block");
            printf (" loop bytes_read = %d]\n", bytes_read);
        }
        /* if nothing is waiting to be read, break out of read loop */
        if ((bytes_read == 0) && (bytes_received == 0)) break;
    }
    if (bytes_received < 0) /* failure */
    {
        if (TRACE && DISPLAYSCREEN)

```

```

    {
        printf ("[get_string_from_tactical_socket receiver block read");
        printf (" failed, bytes_received = %d\n", bytes_received);
    }
}
else if (bytes_received == 0) /* no transfer */
{
    if (TRACE && DISPLAYSCREEN)
    {
        printf("[get_string_from_tactical_socket received 0 bytes!!]\n");
    }
}
else if (bytes_received > 0) /* success */
{
    if (TRACE && DISPLAYSCREEN)
    {
        printf("[get_string_from_tactical_socket received %d bytes]\n",
            bytes_received);
    }
}

/* Check termination *****/
if (strncmp (command_buffer, "shutdown", 8) == 0)
{
    if (TRACE && DISPLAYSCREEN) printf
        ("[get_data_on_tactical_socket: shutdown_signal_received]\n");
    shutdown_tactical_socket ();
}
if (TRACE && DISPLAYSCREEN)
    printf ("[get_string_from_tactical_socket finish]\n");

return;
} /* end get_string_from_tactical_socket () */

/*****/
void record_data_on ()
{
    if (TRACE && DISPLAYSCREEN) printf ("[start record_data_on ()]\n");

    /* Open files for writing */

    if ((TACTICALPARSE) || (TACTICAL == FALSE))
    if ((auvdatafile = fopen (AUVDATAFILENAME, "w")) == NULL)
    {
        printf("record_data_on () unable to open output file %s for writing.",
            AUVDATAFILENAME);
        printf
            ("                Check ownership permissions in current directory.\n");
        printf("Exit.\n");
        exit (-1);
    }
    if (TRACE && DISPLAYSCREEN && (auvdatafile != NULL))
        printf ("[auvdatafile %s open, pointer = %x]\n",
            AUVDATAFILENAME, auvdatafile);

    if (SONARINSTALLED)
    {
        if (((st1000datafile = fopen (ST1000DATAFILE, "w")) == NULL) ||
            ((targetdatafile = fopen (TARGETDATAFILE, "w")) == NULL))
        {
            printf("record_data_on () unable to open st1000 data file %s or ",
                ST1000DATAFILE);
            printf("target data file %s for writing\n", TARGETDATAFILE);
            printf
                ("                Check ownership permissions in current directory.\n");
        }
    }
}

```

```

        printf("Exit.\n");
        exit (-1);
    }
    else
    {
        printf("[st1000datafile %s open, pointer = %x]\n",
               ST1000DATAFILE, st1000datafile);
        printf("[targetdatafile %s open, pointer = %x]\n",
               TARGETDATAFILE, targetdatafile);
    }
}

if ((TACTICALPARSE) || (TACTICAL == FALSE) || (auvdatafile != NULL))
{
    fprintf (auvdatafile, "# auvdatafile %s shows %d ",
            AUVDATAFILENAME, STATEVECTORSIZE);
    fprintf (auvdatafile, "state vector variables at %3.1f intervals.\n\n",
            dt);

    fprintf (auvdatafile,
            "# state                                paddle ");
    fprintf (auvdatafile, " ");
    fprintf (auvdatafile, "phi theta psi ");
    fprintf (auvdatafile, "delta delta port stbd ");

    fprintf (auvdatafile, "bow_ stern bow_ stern ");
    fprintf (auvdatafile, "_ST1000 sonar_ ");
    fprintf (auvdatafile, "_ST725 sonar_ ");
    fprintf (auvdatafile, "Dive Dive");
    fprintf (auvdatafile, "\n");

    fprintf (auvdatafile,
            "# vector t      x      y      z      phi      theta psi      speed ");
    fprintf (auvdatafile, "u      v      w      p      q      r ");
    fprintf (auvdatafile, "x_dot y_dot z_dot _dot _dot _dot ");
    fprintf (auvdatafile, "rudder plane rpm      rpm ");

    fprintf (auvdatafile, "vrtcl vrtcl latrl latrl ");
    fprintf (auvdatafile, "bng range dB ");
    fprintf (auvdatafile, "bng range dB ");
    fprintf (auvdatafile, "Trk1 Trk2");
    fprintf (auvdatafile, "\n\n");
}

if ((auvtextfile = fopen (AUVTEXTFILENAME, "w")) == NULL)
{
    printf("record_data_on () unable to open output file %s for writing. ",
            AUVTEXTFILENAME);
    printf
    ("      Check ownership permissions in current directory.\n");
    printf("Exit.\n");
    exit (-1);
}

if (TRACE && DISPLAYSCREEN)
    printf ("[auvtextfile %s open, pointer = %x]\n",
            AUVTEXTFILENAME, auvtextfile);

fprintf (auvtextfile, "# auvtextfile %s shows %d ",
        AUVDATAFILENAME, STATEVECTORSIZE);
fprintf (auvtextfile, "state vector variables at %3.1f intervals.\n\n",
        dt);

fprintf (auvtextfile,
        "# state                                paddle ");
fprintf (auvtextfile, " ");
fprintf (auvtextfile, "phi theta psi ");
fprintf (auvtextfile, "delta delta port stbd ");

```

```

fprintf (auvtextfile, "bow_ stern bow_ stern ");
fprintf (auvtextfile, "_ST1000 sonar_ ");
fprintf (auvtextfile, "_ST725 sonar_ ");
fprintf (auvtextfile, "Dive Dive");
fprintf (auvtextfile, "\n");

fprintf (auvtextfile,
"# vector t x y z phi theta psi speed ");
fprintf (auvtextfile, "u v w p q r ");
fprintf (auvtextfile, "x_dot y_dot z_dot _dot _dot _dot ");
fprintf (auvtextfile, "rudder plane rpm rpm ");

fprintf (auvtextfile, "vrtcl vrtcl latrl latrl ");
fprintf (auvtextfile, "bng range dB ");
fprintf (auvtextfile, "bng range dB ");
fprintf (auvtextfile, "Trk1 Trk2");
fprintf (auvtextfile, "\n\n");

if (LOOPFOREVER)
    fprintf (auvtextfile, "# Mission replication %d\n",
            replication_count);

/* testing code from wr2t1.c, not currently in use */
/* serial.d is a telemetry test file to check connectivity */
/* if ((serialtestfile = fopen ("serial.d", "r")) <= 0)
{
    printf("record_data_on () can't open test file serial.d\n");
    printf("Exit.\n");
    exit (-1);
}
*/
if (TRACE && DISPLAYSCREEN) printf ("[finish record_data_on ()]\n");

return;
}

/*****
void record_data_off ()
{
    if (TRACE && DISPLAYSCREEN) printf ("[start record_data_off ()]\n");

    if ((auvdatafile != NULL) && TRACE && DISPLAYSCREEN)
    {
        printf ("[flushing and closing auvdatafile %s %x]\n",
                AUVDATAFILENAME, auvdatafile);
        fflush (stdout); /* force completion of screen write */
    }
    if (auvdatafile != NULL)
    {
        if (TRACE && DISPLAYSCREEN) printf ("[auvdatafile flushed]\n");
        fflush (auvdatafile); /* force completion of screen write */
        fflush (auvdatafile); /* force completion of file write */
        fclose (auvdatafile);
        if (TRACE && DISPLAYSCREEN) printf ("[auvdatafile closed]\n");
        fflush (stdout); /* force completion of screen write */
    }
    else if ((TRACE && DISPLAYSCREEN) &&
            ((TACTICAL == FALSE) || (LOCATIONLAB)))
        printf ("[auvdatafile was not open!!]\n");

    if (st1000datafile != NULL)
    {
        if (TRACE && DISPLAYSCREEN) printf ("[st1000datafile flushed]\n");
        fflush (stdout); /* force completion of screen write */
        fflush (st1000datafile); /* force completion of file write */
        fclose (st1000datafile);
    }
}
*****/

```

```

        fflush (targetdatafile);
        fclose (targetdatafile);
        if (TRACE && DISPLAYSCREEN)
            printf ("[st1000datafile and targetdatafile closed]\n");
        fflush (stdout); /* force completion of screen write */
    }

    if (TRACE && DISPLAYSCREEN)
    {
        printf ("[flushing and closing auvtextfile %s %x]\n",
                AUVTEXTFILENAME, auvtextfile);
        fflush (stdout); /* force completion of screen write */
    }
    if (auvtextfile != NULL)
    {
        if (TRACE && DISPLAYSCREEN) printf ("[auvtextfile flushed]\n");
        fflush (stdout); /* force completion of screen write */
        fflush (auvtextfile); /* force completion of file write */
        fclose (auvtextfile);
        if (TRACE && DISPLAYSCREEN) printf ("[auvtextfile closed]\n");
        fflush (stdout); /* force completion of screen write */
    }
    else if (TRACE && DISPLAYSCREEN)
        printf ("[auvtextfile was not open!!]\n");

/*    fclose (serialtestfile); /* serial port test file */

    if (TRACE && DISPLAYSCREEN)
    {
        printf ("[finish record_data_off ()]\n");
        fflush (stdout); /* force completion of screen write */
    }
    return;
}

/*****

int detect_death_spiral ()
{
    static int    turn_direction = 0;
    static double psi_old       = 0.0;
    static double cumulative_turn = 0.0;

    if (TRACE && DISPLAYSCREEN) printf ("[start detect_death_spiral ()]\n");

    /* reset static variables; don't check for spiral */
    if (DEATH_SPIRAL_RESET)
    {
        turn_direction      = 0;
        cumulative_turn     = 0.0;
        DEATH_SPIRAL_RESET = FALSE;
        if (TRACE && DISPLAYSCREEN)
            printf ("[finish detect_death_spiral ()]\n");
        return (FALSE);
    }

    /* Turn direction changed, reset static variables, or not at depth yet */
    if ((dsign (psi_dot) != turn_direction) || (turn_direction == 0) ||
        (fabs (depth_error) > standoff_distance))
    {
        turn_direction = dsign (psi_dot);
        cumulative_turn = 0.0;
        psi_old = psi;
        if (TRACE && DISPLAYSCREEN)
            printf ("[finish detect_death_spiral ()]\n");
        return (FALSE);
    }
}

```



```

    }

    /* Same turn direction, check for full circle */
    cumulative_turn += normalize2 (psi - psi_old);
    psi_old = psi;

    /* Full 360 Degree Constant Direction Turn Means Death Spiral */
    if ((cumulative_turn >= 360) || (cumulative_turn <= -360))
    {
        if (TRACE && DISPLAYSCREEN)
        {
            printf ("[Death Spiral Detected]\n");
            printf ("[finish detect_death_spiral ()]\n");
        }
        return (TRUE);
    }

    /* No Spiral Detected */
    if (TRACE && DISPLAYSCREEN) printf ("[finish detect_death_spiral ()]\n");
    return (FALSE);
} /* end int detect_death_spiral () */

/*****

void cage_dg () /* dg = directional gyro */
{
    /* Low TRUE Logic */
    /* Setting (Cage DG) Low and (UnCage DG) High will Cage the DG */
    via0b_reg = via0b_reg & 0xFE; /* Set bits PB0 Low retaining */
                                   /* other bits */
    via0[ORB_IRB] = via0b_reg;

    /*via0b_reg = via0b_reg | 0x02;*/ /* Set bit PB1 High retaining */
                                   /* other bits */
    /* Using PB3 Pin 44/19 since 48/23 crapped out */
    via0b_reg = via0b_reg | 0x08; /* Set bit PB3 High retaining */
    via0[ORB_IRB] = via0b_reg;

    if (DISPLAYSCREEN) printf("Waiting 20 sec. for Gyro to Cage\n");
    tsleep(2000); /* Wait 20 seconds MAX for Caging */

    return;
} /* end cage_dg () */

/*****

void uncage_dg () /* dg = directional gyro */
{
    /* Low TRUE Logic */
    /* Setting (Cage DG) Hi and (UnCage DG) Low will UnCage the DG */
    via0b_reg = via0b_reg | 0x01; /* Set bit PB0 High retaining */
                                   /* other bits */
    via0[ORB_IRB] = via0b_reg;

    /*via0b_reg = via0b_reg & 0xFD;*/ /* Set bits PB1 Low retaining */
                                   /* other bits */
    /* Using PB3 Pin 44/19 since 48/23 crapped out */
    via0b_reg = via0b_reg & 0xF7; /* Set bits PB3 Low retaining */
    via0[ORB_IRB] = via0b_reg;

```

```

        tsleep(100); /* Wait 1 second for UnCaging */
        return;
    } /* end uncage_dg () */

/*****
double dsign (value)
    double value;
{
    if ((value == 0.0) || (value == -0.0)) return ( 1.0);
    if (value > 0.0) return ( 1.0);
    if (value < 0.0) return (-1.0);
}

*****/

double dtanh(value)
    double value;
{
    if(fabs(value) > 1.0)
    {
        return(dsign(value));
    }
    else
    {
        return(value);
    }

/*    double epv,emv;
    epv = exp(value);
    emv = exp(-value);
    return( (epv - emv)/(epv + emv) );
*/
}

/*****/
/*****/
/* end of external_functions.c */
/*****/
/*****/

```

```

/*****
/*
Program:      parse_functions.c

Authors:      Don Brutzman, Mike Burns, Brad Leonhardt, Duane Davis

Revised:      2 August 96

System:       AUV Gespac 68020/68030, OS-9 version 2.4
Compiler:     Gespac cc Kernighan & Richie (K&R) C

Compilation:  ftp>      put parse_functions.c
               auvsim1> chd execution
               [68020]  auvsim1> make -k2f execution
               [68030]  auvsim1> make      execution

               [Irix]   fletch> make execution

Purpose:      Reduce size of execution.c to allow OS-9 C compiler to work
               Parse command line, mission script commands, control constants

Bugs:         Bizarre vehicle control where thrusters alternate directions
               with no apparent progress is a symptom of incorrect control
               constant initialization.  Corrected.

Note that %F double formats are used instead of %lf on scanf() and sscanf()
calls for OS-9 compatibility.  SGI C compiler does not complain.
gcc on Sun does fail at run-time, so ifdef's are used to support
the proper format string.  (Curse you OS-9! :)

*/

/*****
/* parse_functions.c */

#include "globals.h"
#include "statevector.h"
#include "defines.h"

/*****

void      parse_command_line_flags      ();
int       parse_mission_script_commands ();
void      parse_mission_string_commands ();

void      print_valid_keywords          ();

void      get_control_constants         ();

extern int detect_death_spiral          ();
extern void send_buffer_to_virtual_world_socket ();
extern void clamp                      ();
extern double normalize                 ();
extern double normalize2                ();
extern double dsign                     ();

char      new_filename [160];
char      backupcommand [160];

int       system_result;

/*****

void parse_command_line_flags (argc, argv)
{
    int argc; char **argv;    /* command line arguments */

```

```

int index;
if (DISPLAYSCREEN)
{
    printf ("\n[parse_command_line_flags start: # arguments = %d]\n[",
            argc);
    for (i = 0; i < argc; i++) printf (" %s", argv[i]);
    printf (" ]\n");
}

if (DISPLAYSCREEN) printf ("[parse arguments: ");
{
    for (i = 1; i < argc; i++)
    {
        printf ("%s ", argv[i]);

        if ((strcmp (argv[i-1], "TELEMETRY") != 0) &&
            (strcmp (argv[i-1], "TELEMETRY-FILE") != 0) &&
            (strcmp (argv[i-1], "TELEMETRYFILE") != 0) &&
            (strcmp (argv[i-1], "MISSION") != 0) &&
            (strcmp (argv[i-1], "SCRIPT") != 0) &&
            (strcmp (argv[i-1], "FILE") != 0) &&
            (strcmp (argv[i-1], "FILENAME") != 0))
            for (index = 0; index <= (int)strlen (argv[i]); index++)/*uppercase*/
                argv[i][index] = toupper(argv[i][index]);
    }
    printf ("]\n");
}

strcpy (buffer, ""); /* initialize for SILENT */

for (i = 1; i < argc; i++)
{
    if
        ((strcmp (argv[i], "HELP") == 0) ||
         (strcmp (argv[i], "?") == 0) ||
         (strcmp (argv[i], "/?") == 0) ||
         (strcmp (argv[i], "-?") == 0))
    {
        if (TRACE && DISPLAYSCREEN) printf ("[print_help] ");
        print_help = TRUE;
    }
    else if ((strcmp (argv[i], "KEYBOARD") == 0) ||
             (strcmp (argv[i], "KEY-BOARD") == 0) ||
             (strcmp (argv[i], "KEYBOARD-INPUT") == 0) ||
             (strcmp (argv[i], "KEYBOARDINPUT") == 0))
    {
        if (TRACE && DISPLAYSCREEN) printf ("[KEYBOARDINPUT = TRUE] ");
        KEYBOARDINPUT = TRUE;
    }
    else if (strcmp (argv[i], "TRACE") == 0)
    {
        if (TRACE && DISPLAYSCREEN) printf ("[TRACE = TRUE] ");
        TRACE = TRUE;
    }
    else if ((strcmp (argv[i], "TRACEOFF") == 0) ||
             (strcmp (argv[i], "TRACE-OFF") == 0) ||
             (strcmp (argv[i], "NOTRACE") == 0) ||
             (strcmp (argv[i], "NO-TRACE") == 0))
    {
        if (TRACE && DISPLAYSCREEN) printf ("[TRACE = FALSE] ");
        TRACE = FALSE;
    }
    else if ((strcmp (argv[i], "LOOPFOREVER") == 0) ||
             (strcmp (argv[i], "LOOP-FOREVER") == 0))
    {
        if (TRACE && DISPLAYSCREEN) printf ("[LOOPFOREVER] ");
        LOOPFOREVER = TRUE;
    }
    else if ((strcmp (argv[i], "LOOPONCE") == 0) ||

```

```

        (strcmp (argv[i], "LOOP-ONCE") == 0))
    {
        if (TRACE && DISPLAYSCREEN) printf ("[LOOPONCE] ");
        LOOPFOREVER = FALSE;
    }
    else if ((strcmp (argv[i], "LOOPFILEBACKUP") == 0) ||
             (strcmp (argv[i], "LOOP-FILE-BACKUP") == 0))
    {
        if (TRACE && DISPLAYSCREEN) printf ("[LOOPFILEBACKUP] ");
        LOOPFILEBACKUP = TRUE;
    }
    else if ((strcmp (argv[i], "ENTERCONTROLCONSTANTS") == 0) ||
             (strcmp (argv[i], "ENTER-CONTROL-CONSTANTS") == 0))
    {
        if (TRACE && DISPLAYSCREEN) printf ("[ENTERCONTROLCONSTANTS] ");
        ENTERCONTROLCONSTANTS = TRUE;
    }
    else if ((strcmp (argv[i], "SHOWCONTROLCONSTANTS") == 0) ||
             (strcmp (argv[i], "SHOW-CONTROL-CONSTANTS") == 0))
    {
        if (DISPLAYSCREEN) printf ("[SHOWCONTROLCONSTANTS] ");
        SHOWCONTROLCONSTANTS = TRUE;
    }
    else if ((strcmp (argv[i], "CONTROLCONSTANTSINPUTFILE") == 0) ||
             (strcmp (argv[i], "CONTROL-CONSTANTS-INPUT-FILE") == 0) ||
             (strcmp (argv[i], "CONTROL-CONSTANTS-FILE") == 0) ||
             (strcmp (argv[i], "CONTROLCONSTANTSFILE") == 0))
    {
        LOADCONTROLCONSTANTS = TRUE;
        if (DISPLAYSCREEN)
        {
            printf ("%s ", argv[i]);
            printf (CONTROLCONSTANTSINPUTNAME);
            printf ("\n");
        }
    }
    else if ((strcmp (argv[i], "TACTICAL") == 0) ||
             (strcmp (argv[i], "TACTICAL-HOST") == 0) ||
             (strcmp (argv[i], "TACTICALHOST") == 0) ||
             (strcmp (argv[i], "STRATEGIC") == 0) ||
             (strcmp (argv[i], "STRATEGICHOST") == 0) ||
             (strcmp (argv[i], "STRATEGIC-HOST") == 0))
    {
        TACTICAL = TRUE;
        i++;
        if (i >= argc) print_help = TRUE;
        else
        {
            KEYBOARDINPUT = FALSE;
            sscanf (argv[i], "%s", tactical_remote_host_name);
            if (TRACE && DISPLAYSCREEN)
                printf ("%s %s (set KEYBOARD-OFF)]",
                        argv[i-1], tactical_remote_host_name);
        }
    }
    else if ((strcmp (argv[i], "NO-TACTICAL") == 0) ||
             (strcmp (argv[i], "TACTICAL-OFF") == 0))
    {
        if (DISPLAYSCREEN) printf ("%s\n", argv[i]);
        TACTICAL = FALSE;
    }
    else if ((strcmp (argv[i], "SONARTRACE") == 0) ||
             (strcmp (argv[i], "SONAR-TRACE") == 0) ||
             (strcmp (argv[i], "SONAR-TRACE-ON") == 0))
    {
        if (TRACE && DISPLAYSCREEN) printf ("[SONARTRACE] ");
        SONARTRACE = TRUE;
    }
}

```

```

else if ((strcmp (argv[i], "SONARTRACEOFF") == 0) ||
         (strcmp (argv[i], "SONAR-TRACE-OFF") == 0))
{
    if (TRACE && DISPLAYSCREEN) printf ("[SONARTRACEOFF] ");
    SONARTRACE = FALSE;
}
else if ((strcmp (argv[i], "SONARINSTALLED") == 0) ||
         (strcmp (argv[i], "SONAR-INSTALLED") == 0))
{
    if (TRACE && DISPLAYSCREEN) printf ("[SONAR-INSTALLED] ");
    SONARINSTALLED = TRUE;
}
else if ((strcmp (argv[i], "NOSONARINSTALLED") == 0) ||
         (strcmp (argv[i], "NO-SONAR-INSTALLED") == 0))
{
    if (TRACE && DISPLAYSCREEN) printf ("[NO-SONAR-INSTALLED] ");
    SONARINSTALLED = FALSE;
}
else if ((strcmp (argv[i], "PARALLELPORTTRACE") == 0) ||
         (strcmp (argv[i], "PARALLEL-PORT-TRACE") == 0))
{
    if (TRACE && DISPLAYSCREEN) printf ("[PARALLELPORTTRACE] ");
    PARALLELPORTTRACE = TRUE;
}
else if ((strcmp (argv[i], "SILENT") == 0) ||
         (strcmp (argv[i], "SILENCE") == 0))
{
    if (TRACE && DISPLAYSCREEN) printf ("[SILENT] ");
    /* send to virtual world after socket is open */
    strcpy (buffer, "SILENT"); /* copy current command to buffer */
}
else if ((strcmp (argv[i], "TIMESTEP") == 0) ||
         (strcmp (argv[i], "TIME-STEP") == 0))
{
    i++;
    if (i >= argc) print_help = TRUE;
    else
    {
#ifdef os9
        sscanf (argv[i], "%lf", &TIMESTEP);
#else
        sscanf (argv[i], "%F", &TIMESTEP);
#endif
        if (TRACE && DISPLAYSCREEN)
            printf("[TIMESTEP %3.2lf]", TIMESTEP);
        if (TIMESTEP > 0.0) dt = TIMESTEP;
        else if (TRACE && DISPLAYSCREEN)
            printf(" illegal TIMESTEP value, ignored.");
        if (TRACE && DISPLAYSCREEN) printf(" [dt = %3.2lf]", dt);
    }
}
else if ((strcmp (argv[i], "VIRTUALHOST") == 0) ||
         (strcmp (argv[i], "VIRTUAL-HOST") == 0) ||
         (strcmp (argv[i], "VIRTUAL") == 0) ||
         (strcmp (argv[i], "REMOTE") == 0) ||
         (strcmp (argv[i], "REMOTEHOST") == 0) ||
         (strcmp (argv[i], "REMOTE-HOST") == 0) ||
         (strcmp (argv[i], "DYNAMIC") == 0) ||
         (strcmp (argv[i], "DYNAMICS") == 0))
{
    i++;
    if (i >= argc) print_help = TRUE;
    else
    {
        sscanf (argv[i], "%s", virtual_world_remote_host_name);
        if (TRACE && DISPLAYSCREEN)
            printf("[VIRTUAL-HOST %s]",
                virtual_world_remote_host_name);
    }
}

```

```

    }
else if ((strcmp (argv[i], "TELEMETRY") == 0) ||
        (strcmp (argv[i], "TELEMETRYFILE") == 0) ||
        (strcmp (argv[i], "TELEMETRY-FILE") == 0))
{
    i++;
    if (i >= argc) print_help = TRUE;
    else
    {
        sscanf (argv[i], "%s", TELEMETRYFILENAME);
        if (telemetry_file = fopen (TELEMETRYFILENAME, "r"))
        {
            if (TRACE && DISPLAYSCREEN)
                printf("[TELEMETRY-FILE %s]",
                    virtual_world_remote_host_name);
            REPLAY = TRUE;
        }
        else printf("[Unable to open telemetry file: %s]\n",
            TELEMETRYFILENAME);
    }
}
else if ((strcmp (argv[i], "REALTIME") == 0) ||
        (strcmp (argv[i], "REAL-TIME") == 0))
{
    if (TRACE && DISPLAYSCREEN) printf ("[REALTIME] ");
    REALTIME = TRUE;
}
else if ((strcmp (argv[i], "NOREALTIME") == 0) ||
        (strcmp (argv[i], "NO-REALTIME") == 0) ||
        (strcmp (argv[i], "NO-REAL-TIME") == 0) ||
        (strcmp (argv[i], "NOWAIT") == 0) ||
        (strcmp (argv[i], "NO-WAIT") == 0) ||
        (strcmp (argv[i], "NOPAUSE") == 0) ||
        (strcmp (argv[i], "NO-PAUSE") == 0))
{
    if (TRACE && DISPLAYSCREEN) printf ("[NOWAIT] ");
    REALTIME = FALSE;
}
else if ((strcmp (argv[i], "EMAIL") == 0) ||
        (strcmp (argv[i], "EMAIL-ON") == 0) ||
        (strcmp (argv[i], "E-MAIL") == 0) ||
        (strcmp (argv[i], "E-MAIL-ON") == 0) ||
        (strcmp (argv[i], "EMAILON") == 0))
{
    if (TRACE && DISPLAYSCREEN) printf ("\n[EMAIL ON] ");
    EMAIL = TRUE;
}
else if ((strcmp (argv[i], "EMAILOFF") == 0) ||
        (strcmp (argv[i], "E-MAILOFF") == 0) ||
        (strcmp (argv[i], "EMAIL-OFF") == 0) ||
        (strcmp (argv[i], "E-MAIL-OFF") == 0) ||
        (strcmp (argv[i], "NO-E-MAIL") == 0) ||
        (strcmp (argv[i], "NO-EMAIL") == 0) ||
        (strcmp (argv[i], "NO-E-MAIL") == 0) ||
        (strcmp (argv[i], "NOEMAIL") == 0))
{
    if (TRACE && DISPLAYSCREEN) printf ("[NO EMAIL]");
    EMAIL = FALSE;
}
else if ((strcmp (argv[i], "LOCATIONLAB") == 0) ||
        (strcmp (argv[i], "LOCATION-LAB") == 0))
{
    LOCATIONLAB = TRUE;
}
else if ((strcmp (argv[i], "TETHER") == 0) ||
        (strcmp (argv[i], "TETHERED") == 0))

```

```

{
    LOCATIONLAB = FALSE;
    DISPLAYSCREEN = TRUE;
    REALTIME = TRUE;
}
else if ((strcmp (argv[i], "UNTETHER") == 0) ||
        (strcmp (argv[i], "UNTETHERED") == 0) ||
        (strcmp (argv[i], "NOTETHER") == 0) ||
        (strcmp (argv[i], "NO-TETHER") == 0))
{
    LOCATIONLAB = FALSE;
    DISPLAYSCREEN = FALSE;
    REALTIME = TRUE;
}
else if ((strcmp (argv[i], "TEXT") == 0) ||
        (strcmp (argv[i], "TEXT-ON") == 0))
{
    DISPLAYSCREEN = TRUE;
}
else if ((strcmp (argv[i], "NOTEXT") == 0) ||
        (strcmp (argv[i], "NO-TEXT") == 0))
{
    DISPLAYSCREEN = FALSE;
}
else if ((strcmp (argv[i], "GYROERROR") == 0) ||
        (strcmp (argv[i], "GYRO-ERROR") == 0) ||
        (strcmp (argv[i], "GYRO_ERROR") == 0))
{
    i++;
    if (i >= argc)
    {
        print_help = TRUE;
        if (DISPLAYSCREEN)
            printf (" warning: invalid GYRO-ERROR command.\n");
    }
    else
    {
#ifndef os9
        sscanf (argv[i], "%lf", gyro_error);
#else
        sscanf (argv[i], "%F", gyro_error);
#endif
        if (TRACE && DISPLAYSCREEN)
            printf("[%s %5.2lf]", argv[i-1], gyro_error);
    }
}
else if ((strcmp (argv[i], "DEPTH-CELL-BIAS") == 0) ||
        (strcmp (argv[i], "DEPTHCELLBIAS") == 0) ||
        (strcmp (argv[i], "DEPTH-CELL-ERROR") == 0) ||
        (strcmp (argv[i], "DEPTHCELLERROR") == 0) ||
        (strcmp (argv[i], "DEPTH-BIAS") == 0) ||
        (strcmp (argv[i], "DEPTHBIAS") == 0) ||
        (strcmp (argv[i], "DEPTH-ERROR") == 0) ||
        (strcmp (argv[i], "DEPTHEROR") == 0))
{
    i++;
    if (i >= argc)
    {
        print_help = TRUE;
        if (DISPLAYSCREEN)
            printf (" warning: invalid DEPTH-CELL-BIAS command.\n");
    }
    else
    {
#ifndef os9
        sscanf (argv[i], "%lf", & depth_cell_bias);
#else
        sscanf (argv[i], "%F", & depth_cell_bias);
#endif
    }
}

```



```

#endif
        if (TRACE && DISPLAYSCREEN)
            printf("[%s %5.2lf]", argv[i-1], depth_cell_bias);
    }
    else if ((strcmp (argv[i], "BENCH") == 0) ||
              (strcmp (argv[i], "BENCH-TEST") == 0) ||
              (strcmp (argv[i], "BENCHTEST") == 0))
    {
        DISPLAYSCREEN = TRUE;
        LOCATIONLAB = FALSE;
        KEYBOARDINPUT = TRUE;
        DIVETRACKER = FALSE;
        EMAIL = FALSE;
        REALTIME = TRUE;
        BENCHTEST = TRUE;
    }
    else if ((strcmp (argv[i], "NOSCRIPT") == 0) ||
              (strcmp (argv[i], "NO-SCRIPT") == 0))
    {
        NOSCRIPT = TRUE;
    }
    else if ((strcmp (argv[i], "MISSION") == 0) ||
              (strcmp (argv[i], "SCRIPT") == 0) ||
              (strcmp (argv[i], "FILE") == 0) ||
              (strcmp (argv[i], "FILENAME") == 0))
    {
        i++;
        if (i >= argc)
        {
            print_help = TRUE;
            if (DISPLAYSCREEN)
                printf (" warning: invalid %s command.\n", argv[i]);
        }
        else
        {
            strcpy (new_filename, argv[i]);
            if (DISPLAYSCREEN)
                printf ("\n[%s %s]\n", argv[i-1], new_filename);
#ifdef os9
            sprintf (backupcommand, "cp %s %s", new_filename,
                    AUVSCRIPTFILENAME);
#else
            sprintf (backupcommand, "copy %s %s", new_filename,
                    AUVSCRIPTFILENAME);
#endif
            if (DISPLAYSCREEN)
                printf ("%s\n", backupcommand);
            system_result = system ( backupcommand);
#ifdef os9
            if (TRACE && DISPLAYSCREEN)
                printf ("system_result = %d\n", system_result);
            if (system_result == 512)
            {
                printf ("system_result = %d\n", system_result);
                printf ("Mission not found. Exit.\n");
                exit (-1);
            }
#endif
        }
        auvscriptfile == NULL; /* force re-read */
    }
    else if ((strcmp (argv[i], "SCANWIDTH") == 0) ||
              (strcmp (argv[i], "SCAN-WIDTH") == 0))
    {
        i++;
        if (i >= argc) print_help = TRUE;
        else

```

```

        {
#ifdef os9
            sscanf (argv[i], "%lf", &SCANWIDTH);
#else
            sscanf (argv[i], "%F", &SCANWIDTH);
#endif
            if (DISPLAYSCREEN) printf("[SCANWIDTH %3.2lf]", SCANWIDTH);
            if ((SCANWIDTH <= 0.0) && DISPLAYSCREEN)
            {
                printf(" illegal SCANWIDTH value, ignored.");
                printf(" [SCANWIDTH = %3.2lf]", SCANWIDTH);
            }
        }
        else print_help = TRUE; /*invalid command line entry parameter found */
    } /* end for loop through command line parameters */

    if (print_help) /* print help string *****/
    {
        printf("\nUnreckognized keyword. Usage: execution \n");
        print_valid_keywords ();
        exit (-1);
    }

    if (TRACE && DISPLAYSCREEN)
        printf ("\n[parse_command_line_flags complete]\n");

    return;
} /* end parse_command_line_flags () */

/*****
int parse_mission_script_commands () /* get data from file at program start */
/* mission.script.HELP => descriptions */
{
    /* command_buffer is the string to be parsed */

    int index, read_another_line, parameters_read;
    double parameter1,parameter2,parameter3,parameter4,parameter5,parameter6;
    char parameter_string [60];
    int return_value;

    read_another_line = TRUE;

    /* If Shutdown in Progress, Ignore Commands and Go to Shutdown Script */
    if (HALTSCRIPT)
    {
        return (FALSE);
    }

    /* do not skip to next command in KEYBOARD or script mode until ready */
    if ((t < time_next_command) && (TACTICAL == FALSE)
        && (TACTICALPARSE == FALSE))
    {
        if (TRACE && DISPLAYSCREEN)
        {
            printf ("\n[skip parse_mission_script_commands () until ");
            printf ("t > time_next_command]\n");
        }
        return (FALSE);
    }

    if (TRACE && DISPLAYSCREEN)
        printf ("\n[start parse_mission_script_commands ()]\n");

    if ((GPSFIXINPROGRESS) && (t >= time_postgps_dive))

```

```

{
    if (TACTICAL) /* execution tell tactical gps-fix done */
    {
        if (TRACE && DISPLAYSCREEN)
            printf
            ("\\n[send_buffer_to_tactical_socket (STABLE GPS TIMEOUT)]");
        strcpy (buffer, "STABLE GPS TIMEOUT");
        send_buffer_to_tactical_socket (); /* message */
    }
    GPSFIXINPROGRESS = FALSE;
    read_another_line = FALSE;
}
if ((GPSFIXINPROGRESS) && (t >= time_gps_complete) &&
    (t < time_postgps_dive))
{
    z_command = previous_z_command;
    time_postgps_dive = t + 30.0; /* head back to ordered depth */
    time_next_command = time_postgps_dive;
    time_gps_complete = time_postgps_dive + 1.0;
    read_another_line = FALSE;
    if (DISPLAYSCREEN) printf ("\\n[GPS-FIX complete.]\\n");
}

/* Only look at auvscriptfile if we are in script file execution mode */
if ( (TACTICALPARSE) /* tactical level internal use */
    || (KEYBOARDINPUT) /* execution level */
    || (TACTICAL)) /* execution level getting tactical comms*/
{
    /* no auvscriptfile setup required in these modes */
}
else if ( (auvscriptfile == NULL) /* auvscriptfile not yet opened */
    || feof (auvscriptfile) /* auvscriptfile end-of-file, repeat */
    || auvscriptfilequit) /* flag for all done */
{
    if (DISPLAYSCREEN)
    {
        printf ("\\n[opening a copy of the auvscriptfile %s]\\n",
            AUVSCRIPTFILENAME);
        fflush (stdout);
    }
}

#ifdef os9
    sprintf (backupcommand, "rm %s.backup", AUVSCRIPTFILENAME);
    if (DISPLAYSCREEN) printf ("%s\\n", backupcommand);
    system (backupcommand);
    sprintf (backupcommand, "cp %s %s.backup", AUVSCRIPTFILENAME,
        AUVSCRIPTFILENAME);
    if (DISPLAYSCREEN) printf ("%s\\n", backupcommand);
    system (backupcommand);
#else
    /* OS-9 */
    sprintf (backupcommand, "del %s.backup", AUVSCRIPTFILENAME);
    if (DISPLAYSCREEN) printf ("%s\\n", backupcommand);
    system (backupcommand);
    sprintf (backupcommand, "copy %s %s.backup", AUVSCRIPTFILENAME,
        AUVSCRIPTFILENAME);
    if (DISPLAYSCREEN) printf ("%s\\n", backupcommand);
    system (backupcommand);
#endif

    sprintf (backupcommand, "%s.backup", AUVSCRIPTFILENAME);
    auvscriptfile = fopen (backupcommand, "r"); /* input file */
    if (auvscriptfile == NULL)
    {
        printf ("AUV execution: script file %s\\n", AUVSCRIPTFILENAME);
        printf
        (" (or backup copy %s.backup) not found.\\n",

```

```

        backupcommand);
        printf ("        Ensure you are in the right directory:\n");
        printf ("        auvsim1> chd /h0/execution      or\n");
        printf ("        unix>   cd ~brutzman/execution\n");
        printf
("        Otherwise ensure you have a %s file.\n",
        AUVSCRIPTFILENAME);
        printf ("Exit.\n");
        exit (-1);
    }
    auvscriptfilequit = FALSE;
}
else if (TRACE && DISPLAYSCREEN)
    printf ("\n[auvscriptfile checks out as ready...]\n");

if ((TACTICALPARSE == FALSE) && (NOSCRIPT == FALSE))
{
    /* open auvordersfile - - - - - */
    sprintf (buffer, "%s", AUVORDERSFILENAME);
    if (auvordersfile == NULL)
    {
        auvordersfile = fopen (buffer,"w"); /* output file */

        if (TRACE && DISPLAYSCREEN)
            printf ("\n[auvordersfile = %x, opened successfully]\n",
                auvordersfile);

        fprintf (auvordersfile, "\n\n");
        fprintf (auvordersfile,
            "# NPS AUV file %s: commanded propulsion orders versus time\n",
            AUVORDERSFILENAME);
        fprintf (auvordersfile, "#\n");
        fprintf (auvordersfile, "#          timestep:  %4.2f seconds\n", dt);
        fprintf (auvordersfile, "#\n");
        fprintf (auvordersfile,
            "# time heading North East Depth    rpm    rpm    stern  stern  vertical\n");
        fprintf (auvordersfile,
            "#          x      y      z      port  stbd   plane  rudder  thrusters\n");
        fprintf (auvordersfile,
            "#          bow/stern  bow/stern\n");
        fprintf (auvordersfile, "\n");
    }
    else if (TRACE && DISPLAYSCREEN)
        printf ("auvordersfile (%s) = %x\n", AUVORDERSFILENAME,
            auvordersfile);
    if ((auvordersfile == NULL) && (NOSCRIPT == FALSE))
    {
        printf ("AUV execution:  %s file open unsuccessful.\n", buffer);
        printf ("Error.\n");
        printf ("Exit.\n");
        exit (-1);
    }
}

while (read_another_line) /* ***** Parse loop ***** */
{
    parameter1 = 0.0;
    parameter2 = 0.0;
    parameter3 = 0.0;

    /* Four-way switch: tactical level parses commands internally, or
    /* ~~~~~~ execution level parses commands from
    /* keyboard | tactical ood | mission.script file */

```

```

/* each option gets the next order and puts it in command_buffer */
if (TACTICALPARSE) /* tactical level internal use */
{
    /* command_buffer is already sent and ready */
    read_another_line = FALSE;
}
else if (KEYBOARDINPUT) /* this blocks! */
{
    strcpy (buffer, "Enter command");
    /* send_buffer_to_virtual_world_socket (); /* buffer msg sent */
    printf ("\n%s *** HERE ***: ", buffer);
    strcpy (command_buffer, "");
    gets (command_buffer);
}
else if (TACTICAL) /* execution level getting tactical comms */
{
    /* get command_buffer string from tactical level, nonblocking */
    /* initialize sonar data in case it is generated by tactical*/
    if (TRACE && DISPLAYSCREEN)
        printf ("\n RECEIVE TACTICAL COMMAND *** HERE ***\n");
    strcpy (command_buffer, "");
    get_string_from_tactical_socket ();
    if (strlen (command_buffer) == 0) /* no command was received */
    {
        time_next_command = t + dt; /* same as STEP command */
        read_another_line = FALSE; /* (prevent blocking) */
        if (TRACE && DISPLAYSCREEN)
            printf("no tactical command received, STEP & recheck\n");
        break;
    }
}
else /* get command_buffer string from auvscriptfile */
{
    strcpy (command_buffer, "");
    fgets (command_buffer, 120, auvscriptfile);

    if (feof (auvscriptfile))
    {
        if (DISPLAYSCREEN)
        {
            printf ("\n[EOF condition: (");
            printf ("%s copy) %s.backup, file closed]\n",
                AUVSCRIPTFILENAME, AUVSCRIPTFILENAME);
        }
        fclose (auvscriptfile);
        auvscriptfilequit = TRUE;
        read_another_line = FALSE;
        end_test = TRUE;
        strcpy (command_buffer, "");
        break;
    }
}

/* parse the command, if any ----- */
if ((int)(strlen (command_buffer) <= 120) && TRACE && DISPLAYSCREEN)
{
    printf ("strlen (command_buffer) = %d", strlen (command_buffer));
    printf (">>>%s<<<", command_buffer);
}

parameters_read = sscanf (command_buffer, "%s", keyword);

```

```

if (TRACE && DISPLAYSCREEN)
{
    printf ("parameters_read = %d, keyword = %s",
            parameters_read, keyword);
}

for (index=0; index<=(int)strlen (keyword); index++)/* set uppercase */
    keyword [index] = toupper (keyword [index]);

audible_command = TRUE;

if (TRACE && DISPLAYSCREEN)
{
    printf ("", uppercase keyword = %s\n", keyword);
}

if      ((parameters_read != 1)           ||
         (strlen (keyword)               == 0) ||
         (strlen (command_buffer)        == 0) ||
         (command_buffer [0]             == '\n'))           /* blank line */
{
    audible_command = FALSE;
    read_another_line = TRUE;
    if (DISPLAYSCREEN) printf ("\n");
}
else if ( keyword [0] == '#' )                /* comment */
{
    if (DISPLAYSCREEN) printf ("%s", command_buffer);
    command_buffer [0] = ' ';
}
else if (((keyword [0] == '/') && (keyword [1] == '/')) ||
         ((keyword [0] == '/') && (keyword [1] == '*'))) /* comment */
{
    if (DISPLAYSCREEN) printf ("%s", command_buffer);
    command_buffer [0] = ' ';
    command_buffer [1] = ' ';
}
else if ((strcmp (keyword, "HELP") == 0) ||
         (strcmp (keyword, "?") == 0) ||
         (strcmp (keyword, "-?") == 0) ||
         (strcmp (keyword, "/?") == 0))
{
    if (DISPLAYSCREEN) printf ("\n[HELP] ");
    print_valid_keywords ();
}
else if ((strcmp (keyword, "SONAR_725") == 0) ||
         (strcmp (keyword, "SONAR725") == 0) ||
         (strcmp (keyword, "SONAR-725") == 0))
{
#ifdef os9
    parameters_read = sscanf (command_buffer, "%s%lf%lf%lf",
                              keyword, &parameter1, &parameter2,
                              &parameter3);
#else
    parameters_read = sscanf (command_buffer, "%s%F%F%F",
                              keyword, &parameter1, &parameter2,
                              &parameter3);
#endif

    if (DISPLAYSCREEN && LOCATIONLAB)
        printf ("\n[%s   %6.2f %6.2f %6.2f]\n", keyword, parameter1,
                parameter2, parameter3);
    AUV_ST725_bearing = normalize (parameter1); /* controlled by
AUV */
    if (LOCATIONLAB == FALSE)
    { /*only virtual world provides sonar values when out of water */
        AUV_ST725_range = parameter2;
        AUV_ST725_strength = parameter3;
    }
}

```

```

    }
    else if ((strcmp (keyword, "SONAR_1000") == 0) ||
             (strcmp (keyword, "SONAR1000") == 0) ||
             (strcmp (keyword, "SONAR-1000") == 0))
    {
#ifdef os9
        parameters_read = sscanf (command_buffer, "%s%lf",
                                   keyword, &parameter1);
#else
        parameters_read = sscanf (command_buffer, "%s%F",
                                   keyword, &parameter1);
#endif
        if (DISPLAYSCREEN && LOCATIONLAB)
            printf ("\n[%s %6.2f]\n", keyword, parameter1);
        SONARSCANMODE = 5; /* manual control */
        st1000_command = SONARHEADINGSTEP *
            ((int) (normalize(parameter1) / SONARHEADINGSTEP));
    }

    else if ((strcmp (keyword, "POSITION") == 0) ||
             (strcmp (keyword, "LOCATION") == 0) ||
             (strcmp (keyword, "FIX") == 0))
    /* note this command must be sent to virtual world (AUVsocket.C tests) */
    {
#ifdef os9
        parameters_read = sscanf (command_buffer, "%s%lf%lf%lf",
                                   keyword, &parameter1,
                                   &parameter2, &parameter3);
#else
        parameters_read = sscanf (command_buffer, "%s%F%F%F",
                                   keyword, &parameter1,
                                   &parameter2, &parameter3);
#endif

        if (parameters_read == 4)
        {
            x = parameter1;
            y = parameter2;
            z = parameter3; /* note depth cell will likely update z */
            /* skip line in telemetry file to break point-to-point lines */
            if ((TACTICALPARSE) || (TACTICAL == FALSE))
                fprintf (auvdatafile, "\n");

            if (DISPLAYSCREEN)
                printf ("\n[%s %6.2f %6.2f %6.2f]\n", keyword,
                    parameter1, parameter2, parameter3);
            if (TRACE && DISPLAYSCREEN)
                printf ("\nsending fix to virtual world: [%s]\n",
                    buffer);

            strcpy (buffer, command_buffer); /* copy command to buffer*/
            send_buffer_to_virtual_world_socket (); /* send to vw */
        }
        else if (parameters_read == 3)
        {
            x = parameter1;
            y = parameter2;
            /* skip line in telemetry file to break point-to-point lines */
            if ((TACTICALPARSE) || (TACTICAL == FALSE))
                fprintf (auvdatafile, "\n");

            if (DISPLAYSCREEN)
                printf ("\n[%s %6.2f %6.2f]\n", keyword, parameter1,
                    parameter2);
            if (TRACE && DISPLAYSCREEN)
                printf ("\nsending fix to virtual world: [%s]\n", buffer);
            strcpy (buffer, command_buffer); /* copy command to buffer*/
            send_buffer_to_virtual_world_socket (); /* send to vw */
        }
        else if (DISPLAYSCREEN)

```

```

        printf (" warning: invalid x/y/z fix position, ignored\n");
    }
    else if ((strcmp (keyword, "HOVER") == 0) ||
             (strcmp (keyword, "HOVER-ON") == 0))
    {
#ifdef os9
        parameters_read = sscanf (command_buffer, "%s%lf%lf%lf%lf",
                                   keyword, & parameter1,
                                   & parameter2, & parameter3,
                                   & parameter4, & parameter5);
#else
        parameters_read = sscanf (command_buffer, "%s%F%F%F%F",
                                   keyword, & parameter1,
                                   & parameter2, & parameter3,
                                   & parameter4, & parameter5);
#endif

        if (parameters_read == 6)
        {
            if (DISPLAYSCREEN)
                printf ("\n[%s    %6.2f %6.2f %6.2f %6.2f %6.2f]\n",
                        keyword, parameter1,
                        parameter2, parameter3,
                        parameter4, parameter5);

            HOVERCONTROL      = TRUE;
            REPORTSTABLE      = TRUE;
            WAYPOINTCONTROL   = FALSE;
            ROTATECONTROL     = FALSE;
            LATERALCONTROL    = FALSE;
            THRUSTERCONTROL   = TRUE;
            TARGETCONTROL     = FALSE;
            RECOVERYCONTROL   = FALSE;
            INTEGRALDEPTHCONTROL = FALSE;
            time_int_control_on = t + 10.0; /*give PD 10 seconds */
            SONARSCANMODE     = 1; /* Forward Scan */
            DEADSTICKRUDDER   = TRUE;
            DEATH_SPIRAL_RESET = TRUE;
            rudder_command    = 0.0;
            x_command         = parameter1;
            y_command         = parameter2;
            z_command         = parameter3;
            psi_command       = normalize (parameter4);
            psi_command_hover = psi_command;
            standoff_distance = parameter5;
        }
        else if (parameters_read == 5)
        {
            if (DISPLAYSCREEN)
                printf ("\n[%s    %6.2f %6.2f %6.2f %6.2f]\n",
                        keyword, parameter1,
                        parameter2, parameter3,
                        parameter4);

            HOVERCONTROL      = TRUE;
            REPORTSTABLE      = TRUE;
            WAYPOINTCONTROL   = FALSE;
            ROTATECONTROL     = FALSE;
            LATERALCONTROL    = FALSE;
            TARGETCONTROL     = FALSE;
            RECOVERYCONTROL   = FALSE;
            INTEGRALDEPTHCONTROL = FALSE;
            time_int_control_on = t + 10.0; /* give PD 10 seconds */
            SONARSCANMODE     = 1; /* Forward Scan */
            THRUSTERCONTROL   = TRUE;
            DEADSTICKRUDDER   = TRUE;
            DEATH_SPIRAL_RESET = TRUE;
            rudder_command    = 0.0;
            x_command         = parameter1;
            y_command         = parameter2;
            z_command         = parameter3;
        }
    }
}

```



```

        psi_command      = normalize (parameter4);
        psi_command_hover = psi_command;
    }
    else if (parameters_read == 4)
    {
        if (DISPLAYSCREEN)
            printf ("\n[%s    %6.2f %6.2f %6.2f]\n",
                    keyword, parameter1, parameter2, parameter3);
        HOVERCONTROL      = TRUE;
        REPORTSTABLE      = TRUE;
        WAYPOINTCONTROL    = FALSE;
        ROTATECONTROL      = FALSE;
        LATERALCONTROL     = FALSE;
        TARGETCONTROL      = FALSE;
        RECOVERYCONTROL    = FALSE;
        INTEGRALDEPTHCONTROL = FALSE;
        time_int_control_on = t + 10.0; /* give PD 10 seconds */
        SONARSCANMODE      = 1;        /* Forward Scan */
        THRUSTERCONTROL    = TRUE;
        DEADSTICKRUDDER    = TRUE;
        DEATH_SPIRAL_RESET = TRUE;
        rudder_command     = 0.0;
        x_command          = parameter1;
        y_command          = parameter2;
        z_command          = parameter3;
        psi_command_hover  = psi_command;
    }
    else if (parameters_read == 3)
    {
        if (DISPLAYSCREEN)
            printf ("\n[%s    %6.2f %6.2f]\n", keyword, parameter1,
                    parameter2);
        HOVERCONTROL      = TRUE;
        REPORTSTABLE      = TRUE;
        WAYPOINTCONTROL    = FALSE;
        ROTATECONTROL      = FALSE;
        LATERALCONTROL     = FALSE;
        TARGETCONTROL      = FALSE;
        RECOVERYCONTROL    = FALSE;
        INTEGRALDEPTHCONTROL = FALSE;
        time_int_control_on = t + 10.0; /* give PD 10 seconds */
        SONARSCANMODE      = 1;        /* Forward Scan */
        THRUSTERCONTROL    = TRUE;
        DEADSTICKRUDDER    = TRUE;
        DEATH_SPIRAL_RESET = TRUE;
        rudder_command     = 0.0;
        x_command          = parameter1;
        y_command          = parameter2;
        psi_command_hover  = psi_command;
    }
    else if (parameters_read == 1)
    {
        if (DISPLAYSCREEN) printf ("\n[%s]\n", keyword);
        HOVERCONTROL      = TRUE;
        REPORTSTABLE      = TRUE;
        WAYPOINTCONTROL    = FALSE;
        ROTATECONTROL      = FALSE;
        LATERALCONTROL     = FALSE;
        TARGETCONTROL      = FALSE;
        RECOVERYCONTROL    = FALSE;
        INTEGRALDEPTHCONTROL = FALSE;
        time_int_control_on = t + 10.0; /* give PD 10 seconds */
        SONARSCANMODE      = 1;        /* Forward Scan */
        THRUSTERCONTROL    = TRUE;
        DEADSTICKRUDDER    = TRUE;
        DEATH_SPIRAL_RESET = TRUE;
        rudder_command     = 0.0;
        x_command          = x;        /* stay put! */
    }

```

```

        y_command      = y;
        z_command      = z;
        psi_command_hover = psi; /* "Meet Her" */
    }
    else
    {
        if (DISPLAYSCREEN)
        {
            printf ("\n[%s]\n", keyword);
            printf (" warning: improper number of values, ignored\n");
        }
    }
}
else if ((strcmp (keyword, "GPS")      == 0) ||
         (strcmp (keyword, "GPSFIX")   == 0) ||
         (strcmp (keyword, "GPS-FIX")  == 0) )
{
    if (TACTICALPARSE == FALSE)
    {
        previous_z_command = z_command;
        if (z_command > 40.0) z_command = 41.0; /* deep test tank */
        else z_command = 0.0; /* rapid shallow */
        GPSFIXINPROGRESS = TRUE;
        time_gps_complete = t + 30.0;
        time_postgps_dive = t + 60.0;
        time_next_command = time_gps_complete;

        /* fixed guesstimate of GPS fix interval */
        /* assume GPS-FIX behavior is properly controlled by tactical level */
    }
    if (DISPLAYSCREEN) printf ("\n[GPS-FIX]\n");
}
else if ((GPSFIXINPROGRESS) &&
         ((strcmp (keyword, "GPS-COMPLETE") == 0) ||
          (strcmp (keyword, "GPS-FIX-COMPLETE") == 0) ||
          (strcmp (keyword, "GPSCOMPLETE") == 0) ||
          (strcmp (keyword, "GPSFIXCOMPLETE") == 0)) )
{
    z_command = previous_z_command;
    time_postgps_dive = t + 30.0; /* head back to ordered depth */
    time_next_command = time_postgps_dive;
    time_gps_complete = time_postgps_dive + 1.0;
    read_another_line = FALSE;
    if (DISPLAYSCREEN) printf ("\n[GPS-FIX complete.]\n");
}
else if ((strcmp (keyword, "TARGET-STATION") == 0) ||
         (strcmp (keyword, "TARGETSTATION") == 0))
{
#ifdef os9
    parameters_read = sscanf (command_buffer, "%s%lf%lf%lf%lf%lf",
                              keyword, & parameter1, & parameter2,
                              & parameter3, & parameter4,
                              & parameter5);
#else
    parameters_read = sscanf (command_buffer, "%s%F%F%F%F%F",
                              keyword, & parameter1, & parameter2,
                              & parameter3, & parameter4,
                              & parameter5);
#endif

    if (parameters_read == 3)
    {
        if (DISPLAYSCREEN)
            printf ("\n[%s %6.2f %6.2f]\n", keyword, parameter1,
                parameter2);

        TARGETCONTROL      = TRUE;
        NEWTARGET           = TRUE;
        RECOVERYCONTROL     = FALSE;
        TARGETPOINTING      = TRUE;
    }
}

```

```

        TARGETEDGETRACK      = FALSE;
        HOVERCONTROL         = FALSE;
        WAYPOINTCONTROL      = FALSE;
        FOLLOWWAYPOINTMODE   = FALSE;
        LATERALCONTROL       = FALSE;
        ROTATECONTROL        = FALSE;
        REPORTSTABLE         = TRUE;
        NEWTARGETSTATION     = TRUE;
        target_range_command = parameter1;
        target_bearing_command = normalize (parameter2);
    }
else if (parameters_read == 4)
{
    if (DISPLAYSCREEN)
        printf ("\n[%s    %6.2f %6.2f %6.2f]\n", keyword,
                                parameter1, parameter2,
                                parameter3);

    TARGETCONTROL            = TRUE;
    NEWTARGET                = TRUE;
    RECOVERYCONTROL          = FALSE;
    TARGETPOINTING           = FALSE;
    TARGETEDGETRACK          = FALSE;
    HOVERCONTROL             = FALSE;
    WAYPOINTCONTROL          = FALSE;
    FOLLOWWAYPOINTMODE       = FALSE;
    LATERALCONTROL           = FALSE;
    ROTATECONTROL            = FALSE;
    REPORTSTABLE             = TRUE;
    NEWTARGETSTATION         = TRUE;
    target_range_command     = parameter1;
    target_bearing_command   = normalize (parameter2);
    psi_command              = normalize (parameter3);
}
else if (parameters_read == 5)
{
    if (DISPLAYSCREEN)
        printf ("\n[%s    %6.2f %6.2f %6.2f %6.2f]\n", keyword,
                                parameter1, parameter2,
                                parameter3, parameter4);

    TARGETCONTROL            = TRUE;
    NEWTARGET                = TRUE;
    RECOVERYCONTROL          = FALSE;
    SONARSCANMODE            = 4; /* Locate Target with ST1000 */
    TARGETPOINTING           = TRUE;
    TARGETEDGETRACK          = FALSE;
    HOVERCONTROL             = FALSE;
    WAYPOINTCONTROL          = FALSE;
    FOLLOWWAYPOINTMODE       = FALSE;
    LATERALCONTROL           = FALSE;
    ROTATECONTROL            = FALSE;
    REPORTSTABLE             = TRUE;
    NEWTARGETSTATION         = TRUE;
    target_range_command     = parameter3;
    target_bearing_command   = normalize (parameter4);
    target_range             = parameter1;
    target_bearing           = normalize (parameter2);
    SONARSCANDIRECTION      =
design (normalize2 (target_bearing - normalize (psi + AUV_ST1000_bearing)));
}
else if (parameters_read == 6)
{
    if (DISPLAYSCREEN)
        printf ("\n[%s    %6.2f %6.2f %6.2f %6.2f %6.2f]\n", keyword,
                                parameter1, parameter2,
                                parameter3, parameter4,
                                parameter5);

    TARGETCONTROL            = TRUE;
    NEWTARGET                = TRUE;

```

```

        RECOVERYCONTROL          = FALSE;
        SONARSCANMODE             = 4; /* Locate Target with ST1000 */
        TARGETPOINTING            = FALSE;
        TARGETEDGETRACK           = FALSE;
        HOVERCONTROL              = FALSE;
        WAYPOINTCONTROL           = FALSE;
        FOLLOWWAYPOINTMODE        = FALSE;
        LATERALCONTROL            = FALSE;
        ROTATECONTROL             = FALSE;
        REPORTSTABLE              = TRUE;
        NEWTARGETSTATION          = TRUE;
        target_range_command       = parameter3;
        target_bearing_command     = normalize (parameter4);
        psi_command               = normalize (parameter5);
        target_range              = parameter1;
        target_bearing            = normalize (parameter2);
        SONARSCANDIRECTION        =
design (normalize2 (target_bearing - normalize (psi + AUV_ST1000_bearing)));
    }
    else if ((strcmp (keyword, "EDGE-STATION") == 0) ||
             (strcmp (keyword, "EDGE STATION") == 0))
    {
#ifdef os9
        parameters_read = sscanf (command_buffer, "%s%lf%lf%lf%lf",
                                   keyword, & parameter1, & parameter2,
                                   & parameter3, & parameter4,
                                   & parameter5);
#else
        parameters_read = sscanf (command_buffer, "%s%F%F%F%F",
                                   keyword, & parameter1, & parameter2,
                                   & parameter3, & parameter4,
                                   & parameter5);
#endif

        if (parameters_read == 3)
        {
            if (DISPLAYSCREEN)
                printf ("\n[%s %6.2f %6.2f]\n", keyword, parameter1,
                                                                parameter2);

            TARGETCONTROL          = TRUE;
            NEWTARGET              = TRUE;
            RECOVERYCONTROL        = FALSE;
            TARGETPOINTING         = TRUE;
            TARGETEDGETRACK        = TRUE;
            HOVERCONTROL           = FALSE;
            WAYPOINTCONTROL        = FALSE;
            FOLLOWWAYPOINTMODE     = FALSE;
            LATERALCONTROL         = FALSE;
            ROTATECONTROL          = FALSE;
            REPORTSTABLE           = TRUE;
            NEWTARGETSTATION       = TRUE;
            target_range_command   = parameter1;
            target_bearing_command = normalize (parameter2);
        }
        else if (parameters_read == 4)
        {
            if (DISPLAYSCREEN)
                printf ("\n[%s %6.2f %6.2f %6.2f]\n", keyword,
                                                                parameter1, parameter2,
                                                                parameter3);

            TARGETCONTROL          = TRUE;
            NEWTARGET              = TRUE;
            RECOVERYCONTROL        = FALSE;
            TARGETPOINTING         = FALSE;
            TARGETEDGETRACK        = TRUE;
            HOVERCONTROL           = FALSE;
            WAYPOINTCONTROL        = FALSE;
            FOLLOWWAYPOINTMODE     = FALSE;

```

```

        LATERALCONTROL      = FALSE;
        ROTATECONTROL       = FALSE;
        REPORTSTABLE        = TRUE;
        NEWTARGETSTATION    = TRUE;
        target_range_command = parameter1;
        target_bearing_command = normalize (parameter2);
        psi_command         = normalize (parameter3);
    }
    else if (parameters_read == 5)
    {
        if (DISPLAYSCREEN)
            printf ("\n[%s    %6.2f %6.2f %6.2f %6.2f]\n", keyword,
                    parameter1, parameter2,
                    parameter3, parameter4);

        TARGETCONTROL      = TRUE;
        NEWTARGET           = TRUE;
        RECOVERYCONTROL     = FALSE;
        SONARSCANMODE       = 4; /* Locate Target with ST1000 */
        TARGETPOINTING      = TRUE;
        TARGETEDGETRACK     = TRUE;
        HOVERCONTROL        = FALSE;
        WAYPOINTCONTROL     = FALSE;
        FOLLOWWAYPOINTMODE   = FALSE;
        LATERALCONTROL      = FALSE;
        ROTATECONTROL       = FALSE;
        REPORTSTABLE        = TRUE;
        NEWTARGETSTATION    = TRUE;
        target_range_command = parameter3;
        target_bearing_command = normalize (parameter4);
        target_range        = parameter1;
        target_bearing       = normalize (parameter2);
        SONARSCANDIRECTION  =
    design (normalize2 (target_bearing - normalize (psi + AUV_ST1000_bearing)));
    }
    else if (parameters_read == 6)
    {
        if (DISPLAYSCREEN)
            printf ("\n[%s    %6.2f %6.2f %6.2f %6.2f %6.2f]\n", keyword,
                    parameter1, parameter2,
                    parameter3, parameter4,
                    parameter5);

        TARGETCONTROL      = TRUE;
        NEWTARGET           = TRUE;
        RECOVERYCONTROL     = FALSE;
        SONARSCANMODE       = 4; /* Locate Target with ST1000 */
        TARGETPOINTING      = FALSE;
        TARGETEDGETRACK     = TRUE;
        HOVERCONTROL        = FALSE;
        WAYPOINTCONTROL     = FALSE;
        FOLLOWWAYPOINTMODE   = FALSE;
        LATERALCONTROL      = FALSE;
        ROTATECONTROL       = FALSE;
        REPORTSTABLE        = TRUE;
        NEWTARGETSTATION    = TRUE;
        target_range_command = parameter3;
        target_bearing_command = normalize (parameter4);
        psi_command         = normalize (parameter5);
        target_range        = parameter1;
        target_bearing       = normalize (parameter2);
        SONARSCANDIRECTION  =
    design (normalize2 (target_bearing - normalize (psi + AUV_ST1000_bearing)));
    }
    }
    else if ((strcmp (keyword, "ENTER-TUBE") == 0) ||
             (strcmp (keyword, "ENTERTUBE") == 0))
    {
#ifdef os9
        parameters_read = sscanf (command_buffer, "%s%lf%lf",

```

```

keyword, & parameter1, &parameter2);
#else
parameters_read = sscanf (command_buffer, "%s%F%F",
keyword, & parameter1, &parameter2);
#endif
printf ("%s %5.3lf %5.3lf\n", keyword, parameter1,
parameter2);
SONARSCANMODE = 5;
st1000_command = -SONARHEADINGSTEP *
((int) (normalize(75.0) / SONARHEADINGSTEP));
HOVERCONTROL = FALSE;
WAYPOINTCONTROL = FALSE;
LATERALCONTROL = FALSE;
ROTATECONTROL = FALSE;
TARGETCONTROL = FALSE;
RECOVERYCONTROL = TRUE;
REPORTSTABLE = TRUE;
NEWRECOVERYCOMMAND = TRUE;
range_from_recovery_pt = parameter1;
psi_command = normalize (parameter2);
psi_command_hover = psi_command;
x_command = x;
y_command = y;
/* z_command = z; not needed, use previously ordered value */
}
else if ((strcmp (keyword, "WAIT") == 0) ||
(strcmp (keyword, "RUN") == 0))
{
#ifdef os9
parameters_read = sscanf (command_buffer, "%s%lf",
keyword, & parameter1);
#else
parameters_read = sscanf (command_buffer, "%s%F",
keyword, & parameter1);
#endif
if (DISPLAYSCREEN)
printf ("\n[%s %6.2f; ", keyword, parameter1);
if ((parameters_read == 2) && (parameter1 >= 0.0))
{
if (TACTICALPARSE) return (FALSE);

read_another_line = FALSE;
time_next_command = t + parameter1;
if (DISPLAYSCREEN) printf ("time of next command %6.2f]\n",
time_next_command);
}
else if (DISPLAYSCREEN)
printf (" warning: illegal time value, ignored\n");
}
else if ((strcmp (keyword, "TIME") == 0) ||
(strcmp (keyword, "WAITUNTIL") == 0) ||
(strcmp (keyword, "PAUSEUNTIL") == 0))
{
#ifdef os9
parameters_read = sscanf (command_buffer, "%s%lf",
keyword, & parameter1);
#else
parameters_read = sscanf (command_buffer, "%s%F",
keyword, & parameter1);
#endif
if (DISPLAYSCREEN) printf ("\n[%s %6.2f]\n", keyword,
parameter1);
if (parameters_read == 2)
{
if (TACTICALPARSE) return (FALSE);

read_another_line = FALSE;

```

```

time_next_command = parameter1;
if (parameter1 <= t)
{
    t = parameter1;
    if (DISPLAYSCREEN)
    {
        printf (" warning: time value has reset AUV clock,");
        printf (" velocities reset to zero.\n");
    }
    u          = 0.0;
    v          = 0.0;
    w          = 0.0;
    p          = 0.0;
    q          = 0.0;
    r          = 0.0;
    x_dot      = 0.0;
    y_dot      = 0.0;
    z_dot      = 0.0;
    phi_dot    = 0.0;
    theta_dot  = 0.0;
    psi_dot    = 0.0;
    read_another_line = TRUE; /* no PDU */
}
else if (DISPLAYSCREEN)
    printf (" warning: illegal time value, ignored.\n");
}
else if ((strcmp (keyword, "TIMESTEP") == 0) ||
        (strcmp (keyword, "TIME-STEP") == 0)) /* different than STEP */
{
#ifdef os9
    if (sscanf (command_buffer, "%s%lf", keyword, &parameter1) == 2)
#else
    if (sscanf (command_buffer, "%s%F", keyword, &parameter1) == 2)
#endif
    {
        if (TACTICALPARSE) return (FALSE);
        if ((parameter1 > 0.0) && (parameter1 <= 5.0))
        {
            dt = parameter1;
            if (DISPLAYSCREEN)
                printf ("\n[TIMESTEP %6.2f] ", dt);
            if (TACTICALPARSE == FALSE)
                if (NOSCRIPT == FALSE) fprintf (auvordersfile,
                    "# timestep: %4.2f seconds\n", dt);
        }
        else print_help = TRUE;
    }
    else print_help = TRUE;
}
else if ((strcmp (keyword, "PAUSE") == 0) ||
        (strcmp (keyword, "-PAUSE") == 0))
{
    if (DISPLAYSCREEN)
    {
        printf ("\n[PAUSE]\n");
        strcpy (buffer, " Press any key to continue");
        send_buffer_to_virtual_world_socket (); /* buffer msg sent */
        printf ("\n%s *** HERE ***: ", buffer);
        answer = getchar (); /* pause */
    }
}
else if ((strcmp (keyword, "REALTIME") == 0) ||
        (strcmp (keyword, "REAL-TIME") == 0))
{
    if (DISPLAYSCREEN) printf ("\n[REALTIME] ");
    REALTIME = TRUE;
}

```

```

    }
    else if ((strcmp (keyword, "MISSION") == 0) ||
             (strcmp (keyword, "SCRIPT") == 0) ||
             (strcmp (keyword, "FILE") == 0) ||
             (strcmp (keyword, "FILENAME") == 0))
    {
        parameters_read = sscanf (command_buffer, "%s%s",
                                   keyword, new_filename);
        if (parameters_read == 2)
        {
            if (DISPLAYSCREEN)
                printf ("\n[%s %s]\n", keyword, new_filename);
#ifdef os9
            sprintf (backupcommand, "cp %s %s", new_filename,
                    AUVSCRIPTFILENAME);
#else
            sprintf (backupcommand, "copy %s %s", new_filename,
                    AUVSCRIPTFILENAME);
#endif
            if (DISPLAYSCREEN)
                printf ("%s\n", backupcommand);
            system (backupcommand);
            auvscriptfile == NULL; /* force re-read */
        }
        else
        {
            if (DISPLAYSCREEN)
                printf ("\n[%s] warning: no filename present, ignored\n", keyword);
        }
    }
    else if ((strcmp (keyword, "KEYBOARD") == 0) ||
             (strcmp (keyword, "KEYBOARD-ON") == 0) ||
             (strcmp (keyword, "KEYBOARD-INPUT") == 0) ||
             (strcmp (keyword, "KEYBOARDINPUT") == 0))
    {
        if (DISPLAYSCREEN) printf ("\n[%s]\n", keyword);
        KEYBOARDINPUT = TRUE;
    }
    else if ((strcmp (keyword, "KEYBOARD-OFF") == 0) ||
             (strcmp (keyword, "NO-KEYBOARD") == 0))
    {
        if (DISPLAYSCREEN) printf ("\n[%s]\n", keyword);
        KEYBOARDINPUT = FALSE;
    }
    else if ((strcmp (keyword, "NOWAIT") == 0) ||
             (strcmp (keyword, "NO-WAIT") == 0) ||
             (strcmp (keyword, "NOREALTIME") == 0) ||
             (strcmp (keyword, "NO-REALTIME") == 0) ||
             (strcmp (keyword, "NONREALTIME") == 0) ||
             (strcmp (keyword, "NO-PAUSE") == 0) ||
             (strcmp (keyword, "NOPAUSE") == 0))
    {
        if (DISPLAYSCREEN) printf ("\n[%s]\n", keyword);
        REALTIME = FALSE;
    }
    else if ((strcmp (keyword, "ABORT") == 0))
    {
        HALTSCRIPT = TRUE;
        if (DISPLAYSCREEN) printf ("\n[%s]\n", keyword);
    }
    else if ((strcmp (keyword, "QUIT") == 0) ||
             (strcmp (keyword, "STOP") == 0) ||
             (strcmp (keyword, "DONE") == 0) ||
             (strcmp (keyword, "EXIT") == 0) ||
             (strcmp (keyword, "REPEAT") == 0) ||
             (strcmp (keyword, "RESTART") == 0) ||
             (strcmp (keyword, "COMPLETE") == 0))
    {

```



```

        (strcmp (keyword, "KILL") == 0) ||
        (strcmp (keyword, "SHUTDOWN") == 0))
    {
        /* note most of these commands don't reset LOOPFOREVER, except */
        /* KILL/SHUTDOWN which terminate the dynamics model connection */

        if ((strcmp (keyword, "KILL") == 0) ||
            (strcmp (keyword, "SHUTDOWN") == 0))
        {
            LOOPFOREVER = FALSE;
            strcpy (buffer, "KILL");
            send_buffer_to_virtual_world_socket (); /* buffer msg sent */
        }
        else
        {
            strcpy (buffer, "QUIT");
            send_buffer_to_virtual_world_socket (); /* buffer msg sent */
        }

        if (DISPLAYSCREEN) printf ("\n[%s]\n", keyword);
        if (TRACE && DISPLAYSCREEN) printf ("\n[end_test set TRUE]\n");
        end_test = TRUE;
        read_another_line = FALSE;

        if (TACTICALPARSE) return (FALSE);

        fclose (auvscriptfile);
        auvscriptfilequit = TRUE;
        if (DISPLAYSCREEN)
            printf("\n[QUIT condition: (%s backup file) mission.script.backup, file
closed]\n",
                                AUVSCRIPTFILENAME);

        x_dot = 0.0;
        y_dot = 0.0;
        z_dot = 0.0;
        phi_dot = 0.0; /* degrees/sec */
        theta_dot = 0.0; /* degrees/sec */
        psi_dot = 0.0; /* degrees/sec */
        speed = 0.0;
        u = 0.0;
        v = 0.0;
        w = 0.0;
        p = 0.0; /* degrees/sec */
        q = 0.0; /* degrees/sec */
        r = 0.0; /* degrees/sec */
        delta_planes = 0.0; /* degrees */
        delta_rudder = 0.0; /* degrees */
        port_rpm = 0;
        stbd_rpm = 0;
        vertical_thruster_volts = 0.0;
        lateral_thruster_volts = 0.0;
        return (FALSE);
    }
    else if ((strcmp (keyword, "RPM") == 0) ||
             (strcmp (keyword, "SPEED") == 0) ||
             (strcmp (keyword, "PROPS") == 0) ||
             (strcmp (keyword, "PROPELLORS") == 0))
    {
#ifdef os9
        parameters_read = sscanf (command_buffer, "%s%lf%lf",
                                keyword, & parameter1,
                                & parameter2);
#else
        parameters_read = sscanf (command_buffer, "%s%F%F",
                                keyword, & parameter1,
                                & parameter2);
#endif
        if (parameters_read == 3)

```

```

        {
            WAYPOINTCONTROL = FALSE;
            ROTATECONTROL    = FALSE;
            HOVERCONTROL     = FALSE;
            TARGETCONTROL    = FALSE;
            RECOVERYCONTROL  = FALSE;
            SONARSCANMODE    = 1;      /* Forward Scan */
            if (DISPLAYSCREEN)
                printf ("\n[%s    %6.2f %6.2f]\n", keyword, parameter1,
parameter2);
            port_rpm_command = parameter1;
            stbd_rpm_command = parameter2;
        }
    else
    {
#ifdef os9
        parameters_read = sscanf (command_buffer, "%s%lf",
                                   keyword, & parameter1);
#else
        parameters_read = sscanf (command_buffer, "%s%F",
                                   keyword, & parameter1);
#endif

        if (DISPLAYSCREEN)
            printf ("\n[%s    %6.2f]\n", keyword, parameter1);
        if (parameters_read == 2)
        {
            WAYPOINTCONTROL = FALSE;
            ROTATECONTROL    = FALSE;
            HOVERCONTROL     = FALSE;
            TARGETCONTROL    = FALSE;
            RECOVERYCONTROL  = FALSE;
            SONARSCANMODE    = 1;      /* Forward Scan */
            port_rpm_command = parameter1;
            stbd_rpm_command = parameter1;
        }
        else if (DISPLAYSCREEN) printf (" warning: no value, ignored\n");
    }
}
else if ((strcmp (keyword, "COURSE") == 0) ||
         (strcmp (keyword, "HEADING") == 0) ||
         (strcmp (keyword, "YAW") == 0))
{
#ifdef os9
    parameters_read = sscanf (command_buffer, "%s%lf",
                              keyword, & parameter1);
#else
    parameters_read = sscanf (command_buffer, "%s%F",
                              keyword, & parameter1);
#endif

    if (DISPLAYSCREEN) printf ("\n[%s    %6.2f]\n", keyword,
parameter1);
    if (parameters_read == 2)
    {
        DEADSTICKRUDDER = FALSE;
        WAYPOINTCONTROL = FALSE;
        TARGETPOINTING   = FALSE;
        psi_command      = normalize (parameter1);
        psi_command_hover = psi_command;
        rotate_command   = 0.0;
        lateral_command  = 0.0;
        ROTATECONTROL    = FALSE;
        LATERALCONTROL    = FALSE;
        REPORTSTABLE     = TRUE;
    }
    else if (DISPLAYSCREEN) printf (" warning: no value, ignored\n");
}
else if ((strcmp (keyword, "PITCH") == 0) ||
         (strcmp (keyword, "THETA") == 0))

```

```

{
#ifdef os9
    parameters_read = sscanf (command_buffer, "%s%lf",
                                keyword, & parameter1);
#else
    parameters_read = sscanf (command_buffer, "%s%F",
                                keyword, & parameter1);
#endif
    if (DISPLAYSCREEN) printf ("\n[%s    %6.2f]\n", keyword,
parameter1);
    if (parameters_read == 2)
    {
        DEADSTICKRUDDER = FALSE;
        WAYPOINTCONTROL = FALSE;
        TARGETPOINTING = FALSE;
        theta_command = normalize (parameter1);
        rotate_command = 0.0;
        lateral_command = 0.0;
        ROTATECONTROL = FALSE;
        LATERALCONTROL = FALSE;
        REPORTSTABLE = TRUE;
    }
    else if (DISPLAYSCREEN) printf (" warning: no value, ignored\n");
}
else if ((strcmp (keyword, "TURN") == 0) ||
        (strcmp (keyword, "CHANGE-COURSE") == 0))
{
#ifdef os9
    parameters_read = sscanf (command_buffer, "%s%lf",
                                keyword, & parameter1);
#else
    parameters_read = sscanf (command_buffer, "%s%F",
                                keyword, & parameter1);
#endif
    if (DISPLAYSCREEN) printf ("\n[%s    %6.2f]\n", keyword,
parameter1);
    if (parameters_read == 2)
    {
        DEADSTICKRUDDER = FALSE;
        WAYPOINTCONTROL = FALSE;
        ROTATECONTROL = FALSE;
        TARGETPOINTING = FALSE;
        psi_command = normalize (psi_command + parameter1);
    }
    else if (DISPLAYSCREEN) printf (" warning: no value, ignored\n");
}
else if (strcmp (keyword, "RUDDER") == 0)
{
#ifdef os9
    parameters_read = sscanf (command_buffer, "%s%lf",
                                keyword, & parameter1);
#else
    parameters_read = sscanf (command_buffer, "%s%F",
                                keyword, & parameter1);
#endif
    if (DISPLAYSCREEN)
        printf ("\n[%s %6.2f, THRUSTERCONTROL == FALSE]\n", keyword,
parameter1);
    if (parameters_read == 2)
    {
        DEADSTICKRUDDER = TRUE;
        WAYPOINTCONTROL = FALSE;
        ROTATECONTROL = FALSE;
        HOVERCONTROL = FALSE;
        TARGETCONTROL = FALSE;
        RECOVERYCONTROL = FALSE;
        THRUSTERCONTROL = FALSE;
        SONARSCANMODE = 1;      /* Forward Scan */
    }
}

```

```

/* note rudder_command is for forward rudder, negative com-
mand turns left */
    rudder_command = normalize2 (- parameter1);
}
else
{
    if (DISPLAYSCREEN)
        printf (" warning: improper value, rudder order ignored\n");
}
}
else if (strcmp (keyword, "DEADSTICKRUDDER") == 0)
{
#ifdef os9
    parameters_read = sscanf (command_buffer, "%s%lf",
                                keyword, & parameter1);
#else
    parameters_read = sscanf (command_buffer, "%s%F",
                                keyword, & parameter1);
#endif

    if (parameters_read == 2)
    {
        if (DISPLAYSCREEN)
            printf ("\n[%s %6.2f, THRUSTERCONTROL == FALSE]\n",
keyword, parameter1);
        DEADSTICKRUDDER = TRUE;
        WAYPOINTCONTROL = FALSE;
        ROTATECONTROL = FALSE;
        HOVERCONTROL = FALSE;
        TARGETCONTROL = FALSE;
        RECOVERYCONTROL = FALSE;
        THRUSTERCONTROL = FALSE;
        SONARSCANMODE = 1; /* Forward Scan */
        rudder_command = normalize2 (parameter1);
    }
    else
    {
        if (DISPLAYSCREEN) printf ("\n[%s, ] ", keyword);
        DEADSTICKRUDDER = TRUE;
        WAYPOINTCONTROL = FALSE;
        ROTATECONTROL = FALSE;
        rudder_command = 0.0;
        if (DISPLAYSCREEN)
            printf (" warning: improper/missing value, rudder set to 0\n");
    }
}
else if (strcmp (keyword, "DEPTH") == 0)
{
#ifdef os9
    parameters_read = sscanf (command_buffer, "%s%lf",
                                keyword, & parameter1);
#else
    parameters_read = sscanf (command_buffer, "%s%F",
                                keyword, & parameter1);
#endif

    if (DISPLAYSCREEN) printf ("\n[%s %6.2f]\n", keyword,
parameter1);
    if (parameters_read == 2)
    {
        DEADSTICKPLANES = FALSE;
        INTEGRALDEPTHCONTROL = FALSE;
        TARGETCONTROL = FALSE;
        time_int_control_on = t + 10.0; /* give PD 10 seconds */
        z_command = parameter1;

        if (HOVERCONTROL) /* report when stable again */
            REPORTSTABLE = TRUE;
    }
    else if (DISPLAYSCREEN) printf (" warning: no value, ignored\n");
}

```

```

    }
    else if (strcmp (keyword, "PLANES") == 0)
    {
#ifdef os9
        parameters_read = sscanf (command_buffer, "%s%lf",
                                   keyword, & parameter1);
#else
        parameters_read = sscanf (command_buffer, "%s%F",
                                   keyword, & parameter1);
#endif
        if (DISPLAYSCREEN)
            printf ("\n[%s %6.2f, THRUSTERCONTROL == FALSE]\n",
                    keyword, parameter1);
        if (parameters_read == 2)
        {
            DEADSTICKPLANES = TRUE;
            THRUSTERCONTROL = FALSE;
            /* note planes_command is for forward planes, negative command points down */
            planes_command = - parameter1;
        }
        else if (DISPLAYSCREEN)
            printf (" warning: improper value, planes order ignored\n");
    }
    else if (strcmp (keyword, "DEADSTICKPLANES") == 0)
    {
#ifdef os9
        parameters_read = sscanf (command_buffer, "%s%lf",
                                   keyword, & parameter1);
#else
        parameters_read = sscanf (command_buffer, "%s%F",
                                   keyword, & parameter1);
#endif
        if (parameters_read == 2)
        {
            if (DISPLAYSCREEN)
                printf ("\n[%s %6.2f, THRUSTERCONTROL == FALSE]\n",
                        keyword, parameter1);
            DEADSTICKPLANES = TRUE;
            THRUSTERCONTROL = FALSE;
            planes_command = parameter1;
        }
        else
        {
            if (DISPLAYSCREEN) printf ("\n[%s] ", keyword);
            DEADSTICKPLANES = TRUE;
            planes_command = 0.0;
            if (DISPLAYSCREEN)
                printf (" warning: improper value, planes set to 0\n");
        }
    }
    else if ((strcmp (keyword, "THRUSTERS-ON") == 0) ||
             (strcmp (keyword, "THRUSTERS") == 0) ||
             (strcmp (keyword, "THRUSTERON") == 0) ||
             (strcmp (keyword, "THRUSTERSON") == 0))
    {
        if (DISPLAYSCREEN) printf ("\n[%s]\n", keyword);
        THRUSTERCONTROL = TRUE;
    }
    else if ((strcmp (keyword, "NOTHRUSTER") == 0) ||
             (strcmp (keyword, "NOTHRUSTERS") == 0) ||
             (strcmp (keyword, "THRUSTERS-OFF") == 0) ||
             (strcmp (keyword, "THRUSTERSOFF") == 0))
    {
        if (DISPLAYSCREEN) printf ("\n[%s]\n", keyword);
        THRUSTERCONTROL = FALSE;
        ROTATECONTROL = FALSE;
        LATERALCONTROL = FALSE;
        HOVERCONTROL = FALSE;
    }

```

```

        TARGETCONTROL = FALSE;
        RECOVERYCONTROL = FALSE;
        SONARSCANMODE = 1; /* Forward Scan */
    }
    else if (strcmp (keyword, "ROTATE") == 0)
    {
#ifdef os9
        parameters_read = sscanf (command_buffer, "%s%lf",
                                   keyword, & parameter1);
#else
        parameters_read = sscanf (command_buffer, "%s%F",
                                   keyword, & parameter1);
#endif
        if (DISPLAYSCREEN) printf ("\n[%s %6.2f]\n", keyword,
parameter1);
        if (parameters_read == 2)
        {
            THRUSTERCONTROL = TRUE;
            WAYPOINTCONTROL = FALSE;
            HOVERCONTROL = FALSE;
            TARGETCONTROL = FALSE;
            RECOVERYCONTROL = FALSE;
            SONARSCANMODE = 1; /* Forward Scan */
            rotate_command = parameter1;
            clamp (&rotate_command, -12.0, 12.0, "rotate_command");
            lateral_command = 0.0;
            ROTATECONTROL = TRUE;
            LATERALCONTROL = FALSE;
        }
        else if (DISPLAYSCREEN) printf (" warning: no value, ignored\n");
    }
    else if ((strcmp (keyword, "NOROTATE") == 0) ||
              (strcmp (keyword, "ROTATEOFF") == 0) ||
              (strcmp (keyword, "ROTATE-OFF") == 0))
    {
        if (DISPLAYSCREEN) printf ("\n[%s]\n", keyword);
        rotate_command = 0.0;
        ROTATECONTROL = FALSE;
    }
    else if (strcmp (keyword, "LATERAL") == 0)
    {
#ifdef os9
        parameters_read = sscanf (command_buffer, "%s%lf",
                                   keyword, & parameter1);
#else
        parameters_read = sscanf (command_buffer, "%s%F",
                                   keyword, & parameter1);
#endif
        if (DISPLAYSCREEN) printf ("\n[%s %6.2f]\n",
keyword, parameter1);
        if (parameters_read == 2)
        {
            THRUSTERCONTROL = TRUE;
            WAYPOINTCONTROL = FALSE;
            HOVERCONTROL = FALSE;
            TARGETCONTROL = FALSE;
            RECOVERYCONTROL = FALSE;
            SONARSCANMODE = 1; /* Forward Scan */
            rotate_command = 0.0;
            lateral_command = parameter1;
            ROTATECONTROL = FALSE;
            LATERALCONTROL = TRUE;
        }
        else if (DISPLAYSCREEN) printf (" warning: no value, ignored\n");
    }
    else if ((strcmp (keyword, "NOLATERAL") == 0) ||
              (strcmp (keyword, "LATERALOFF") == 0) ||
              (strcmp (keyword, "LATERAL-OFF") == 0))

```

```

    {
        if (DISPLAYSCREEN) printf ("\n[%s]\n", keyword);
        lateral_command = 0.0;
        LATERALCONTROL = FALSE;
    }
    else if ((strcmp (keyword, "DIVETRACKER1") == 0) ||
             (strcmp (keyword, "DIVE-TRACKER1") == 0) ||
             (strcmp (keyword, "DIVE-TRACKER-1") == 0) ||
             (strcmp (keyword, "DIVE_TRACKER1") == 0) ||
             (strcmp (keyword, "DIVE_TRACKER_1") == 0) )
        /* note this command must be sent to virtual world (AUVsocket.C tests) */
        {
#ifdef os9
            parameters_read = sscanf (command_buffer, "%s%lf%lf%lf",
                                     keyword, & parameter1,
                                     & parameter2, & parameter3);
#else
            parameters_read = sscanf (command_buffer, "%s%F%F%F",
                                     keyword, & parameter1,
                                     & parameter2, & parameter3);
#endif

            if (DISPLAYSCREEN)
                printf ("\n[%s] %6.2f %6.2f %6.2f\n",
                        keyword, parameter1,
                        parameter2, parameter3);

            if (parameters_read == 4)
            {
                DiveTracker1_x = parameter1;
                DiveTracker1_y = parameter2;
                DiveTracker1_z = parameter3;

                if (TRACE && DISPLAYSCREEN)
                    printf ("\nsending divetracker data to virtual world: [%s]\n", buffer);
                strcpy (buffer, command_buffer); /* copy command to buffer*/
                send_buffer_to_virtual_world_socket (); /* send to vw */
            }
            else if (DISPLAYSCREEN)
                printf (" warning: invalid DiveTracker1 position, ignored\n");
        }
        else if ((strcmp (keyword, "DIVETRACKER2") == 0) ||
                 (strcmp (keyword, "DIVE-TRACKER2") == 0) ||
                 (strcmp (keyword, "DIVE-TRACKER-2") == 0) ||
                 (strcmp (keyword, "DIVE_TRACKER2") == 0) ||
                 (strcmp (keyword, "DIVE_TRACKER_2") == 0) )
            /* note this command must be sent to virtual world (AUVsocket.C tests) */
            {
#ifdef os9
                parameters_read = sscanf (command_buffer, "%s%lf%lf%lf",
                                         keyword, & parameter1,
                                         & parameter2, & parameter3);
#else
                parameters_read = sscanf (command_buffer, "%s%F%F%F",
                                         keyword, & parameter1,
                                         & parameter2, & parameter3);
#endif

                if (DISPLAYSCREEN)
                    printf ("\n[%s] %6.2f %6.2f %6.2f\n",
                            keyword, parameter1,
                            parameter2, parameter3);

                if (parameters_read == 4)
                {
                    DiveTracker2_x = parameter1;
                    DiveTracker2_y = parameter2;
                    DiveTracker2_z = parameter3;

                    if (TRACE && DISPLAYSCREEN)
                        printf ("\nsending divetracker data to virtual world: [%s]\n", buffer);
                    strcpy (buffer, command_buffer); /* copy command to buffer*/
                }
            }
        }
    }

```

```

        send_buffer_to_virtual_world_socket (); /* send to vw */
    }
    else if (DISPLAYSCREEN)
        printf (" warning: invalid DiveTracker2 position, ignored\n");
}
else if ((strcmp (keyword, "GYROERROR") == 0) ||
        (strcmp (keyword, "GYRO-ERROR") == 0) ||
        (strcmp (keyword, "GYRO_ERROR") == 0))
{
#ifdef os9
    parameters_read = sscanf (command_buffer, "%s%lf",
                               keyword, & parameter1);
#else
    parameters_read = sscanf (command_buffer, "%s%F",
                               keyword, & parameter1);
#endif
    if (DISPLAYSCREEN) printf ("\n[%s %6.2f]\n", keyword,
parameter1);
    if (parameters_read == 2)
    {
        gyro_error = parameter1;
    }
    else if (DISPLAYSCREEN)
        printf (" warning: invalid GYRO-ERROR command, ignored\n");
}
else if ((strcmp (keyword, "DEPTH-CELL-BIAS") == 0) ||
        (strcmp (keyword, "DEPTHCELLBIAS") == 0) ||
        (strcmp (keyword, "DEPTH-CELL-ERROR") == 0) ||
        (strcmp (keyword, "DEPTHCELLERROR") == 0) ||
        (strcmp (keyword, "DEPTH-BIAS") == 0) ||
        (strcmp (keyword, "DEPTHBIAS") == 0) ||
        (strcmp (keyword, "DEPTH-ERROR") == 0) ||
        (strcmp (keyword, "DEPTHEROR") == 0))
{
#ifdef os9
    parameters_read = sscanf (command_buffer, "%s%lf",
                               keyword, & parameter1);
#else
    parameters_read = sscanf (command_buffer, "%s%F",
                               keyword, & parameter1);
#endif
    if (DISPLAYSCREEN) printf ("\n[%s %6.2f]\n", keyword,
parameter1);
    if (parameters_read == 2)
    {
        depth_cell_bias = parameter1;
    }
    else if (DISPLAYSCREEN)
        printf (" warning: invalid DEPTH-CELL-BIAS command, ignored\n");
}
else if ((strcmp (keyword, "ORIENTATION") == 0) ||
        (strcmp (keyword, "ROTATION") == 0) )
{
#ifdef os9
    parameters_read = sscanf (command_buffer, "%s%lf%lf%lf",
                               keyword, & parameter1,
                               & parameter2, & parameter3);
#else
    parameters_read = sscanf (command_buffer, "%s%F%F%F",
                               keyword, & parameter1,
                               & parameter2, & parameter3);
#endif
    if (DISPLAYSCREEN)
        printf ("\n[%s %6.2f %6.2f %6.2f]\n", keyword,
parameter1,
parameter2, parameter3);
    if (parameters_read == 4)
    {

```



```

        phi = parameter1;
        theta = parameter2;
        psi = parameter3;
    }
    else
        if (DISPLAYSCREEN)
            printf (" warning: invalid phi/theta/psi orientation, ignored\n");
    }
    else if (strcmp (keyword, "POSTURE") == 0)
    {
#ifdef os9
        parameters_read = sscanf (command_buffer, "%s%lf%lf%lf%lf%lf",
                                   keyword, & parameter1,
                                   & parameter2, & parameter3,
                                   & parameter4, & parameter5,
                                   & parameter6);
#else
        parameters_read = sscanf (command_buffer, "%s%F%F%F%F%F",
                                   keyword, & parameter1,
                                   & parameter2, & parameter3,
                                   & parameter4, & parameter5,
                                   & parameter6);
#endif

        if (DISPLAYSCREEN)
            printf ("\n[%s %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f]\n",
                    keyword, parameter1,
                    parameter2, parameter3,
                    parameter4, parameter5,
                    parameter6);

        if (parameters_read == 7)
        {
            x = parameter1;
            y = parameter2;
            z = parameter3;
            phi = parameter4;
            theta = parameter5;
            psi = parameter6;
            start_psi = parameter6;
            kal_init_z = TRUE;
            INTEGRALDEPTHCONTROL = FALSE;
            time_int_control_on = t + 10.0; /* give PD 10 seconds */

            /* skip line in telemetry file to break point-to-point lines */
            if ((TACTICAL == FALSE) || (TACTICALPARSE))
                fprintf (auvdatafile, "\n");
        }
        else if (DISPLAYSCREEN)
            printf (" warning: invalid posture values (6 required), ignored\n");
    }
    else if ((strcmp (keyword, "OCEANCURRENT") == 0) ||
             (strcmp (keyword, "OCEAN-CURRENT") == 0))
    {
#ifdef os9
        parameters_read = sscanf (command_buffer, "%s%lf%lf%lf",
                                   keyword, & parameter1,
                                   & parameter2, & parameter3);
#else
        parameters_read = sscanf (command_buffer, "%s%F%F%F",
                                   keyword, & parameter1,
                                   & parameter2, & parameter3);
#endif

        if (parameters_read == 4)
        {
            if (DISPLAYSCREEN)
                printf ("\n[%s %6.2f %6.2f %6.2f]\n",
                        keyword, parameter1,
                        parameter2, parameter3);
            AUV_oceancurrent_x = parameter1;
        }
    }
}

```

```

        AUV_oceancurrent_y = parameter2;
        AUV_oceancurrent_z = parameter3;
    }
    else if (parameters_read == 3)
    {
        if (DISPLAYSCREEN)
            printf ("\n[%s    %6.2f %6.2f]\n", keyword, parameter1,
                parameter2);
        AUV_oceancurrent_x = parameter1;
        AUV_oceancurrent_y = parameter2;
    }
    else
    {
        if (DISPLAYSCREEN)
        {
            printf ("\n warning: improper number of OCEAN-CURRENT ");
            printf ("values, ignored\n");
        }
    }
}
else if ((strcmp (keyword, "CONTINUE") == 0) ||
        (strcmp (keyword, "GO") == 0))
{
    if (DISPLAYSCREEN) printf ("\n[%s]\n", keyword);
    return (FALSE); /* no action required */
}
else if ((strcmp (keyword, "STEP") == 0) ||
        (strcmp (keyword, "SINGLE-STEP") == 0))
{
    if (DISPLAYSCREEN) printf ("\n[%s]\n", keyword);
    time_next_command = t + dt;
    read_another_line = FALSE;
}
else if ((strcmp (keyword, "TRACE") == 0) ||
        (strcmp (keyword, "TRACE-ON") == 0))
{
    if (DISPLAYSCREEN) printf ("\n[TRACE = TRUE] ");
    TRACE = TRUE;
}
else if ((strcmp (keyword, "TRACEOFF") == 0) ||
        (strcmp (keyword, "TRACE-OFF") == 0) ||
        (strcmp (keyword, "NOTRACE") == 0) ||
        (strcmp (keyword, "NO-TRACE") == 0))
{
    if (DISPLAYSCREEN) printf ("\n[TRACE = FALSE] ");
    TRACE = FALSE;
}
#endif
/* save space due to OS9 size problems */
else if ((strcmp (keyword, "LOOPFOREVER") == 0) ||
        (strcmp (keyword, "LOOP-FOREVER") == 0))
{
    if (DISPLAYSCREEN) printf ("\n[LOOPFOREVER] ");
    LOOPFOREVER = TRUE;
}
else if ((strcmp (keyword, "LOOPONCE") == 0) ||
        (strcmp (keyword, "LOOP-ONCE") == 0))
{
    if (DISPLAYSCREEN) printf ("\n[LOOPONCE] ");
    LOOPFOREVER = FALSE;
}
else if ((strcmp (keyword, "LOOPFILEBACKUP") == 0) ||
        (strcmp (keyword, "LOOP-FILE-BACKUP") == 0))
{
    if (DISPLAYSCREEN) printf ("\n[LOOPFILEBACKUP] ");
    LOOPFILEBACKUP = TRUE;
}
else if ((strcmp (keyword, "ENTERCONTROLCONSTANTS") == 0) ||

```

```

        (strcmp (keyword, "ENTER-CONTROL-CONSTANTS") == 0))
    {
        if (DISPLAYSCREEN) printf ("\n[ENTERCONTROLCONSTANTS] ");
        ENTERCONTROLCONSTANTS = TRUE;
        get_control_constants ();
        fflush (stdin);
    }
    else if ((strcmp (keyword, "SHOWCONTROLCONSTANTS") == 0) ||
             (strcmp (keyword, "SHOW-CONTROL-CONSTANTS") == 0))
    {
        if (DISPLAYSCREEN) printf ("\n[SHOWCONTROLCONSTANTS] ");
        SHOWCONTROLCONSTANTS = TRUE;
        get_control_constants ();
        fflush (stdout);
    }
    else if ((strcmp (keyword, "CONTROLCONSTANTSINPUTFILE") == 0) ||
             (strcmp (keyword, "CONTROL-CONSTANTS-INPUT-FILE") == 0) ||
             (strcmp (keyword, "CONTROL-CONSTANTS-FILE") == 0) ||
             (strcmp (keyword, "CONTROLCONSTANTSFILE") == 0))
    {
        LOADCONTROLCONSTANTS = TRUE;
        get_control_constants ();
        if (DISPLAYSCREEN)
            printf ("\n[CONTROLCONSTANTSINPUTFILE %s]",
                    CONTROLCONSTANTSINPUTNAME);
    }
    else if ((strcmp (keyword, "SLIDINGMODECOURSE") == 0) ||
             (strcmp (keyword, "SLIDING-MODE-COURSE") == 0))
    {
        if (DISPLAYSCREEN) printf ("\n[%s = TRUE]\n", keyword);
        SLIDINGMODECOURSE = TRUE;
        WAYPOINTCONTROL = FALSE;
        ROTATECONTROL = FALSE;
        HOVERCONTROL = FALSE;
        TARGETCONTROL = FALSE;
        RECOVERYCONTROL = FALSE;
        SONARSCANMODE = 1; /* Forward Scan */
    }
    else if ((strcmp (keyword, "SLIDINGMODEOFF") == 0) ||
             (strcmp (keyword, "SLIDING-MODE-OFF") == 0))
    {
        if (DISPLAYSCREEN)
            printf ("\n[%s: SLIDINGMODECOURSE = FALSE]\n", keyword);
        SLIDINGMODECOURSE = FALSE;
    }
    else if ((strcmp (keyword, "SONARTRACE") == 0) ||
             (strcmp (keyword, "SONAR-TRACE") == 0) ||
             (strcmp (keyword, "SONAR-TRACE-ON") == 0))
    {
        if (DISPLAYSCREEN) printf ("\n[SONARTRACE] ");
        SONARTRACE = TRUE;
    }
    else if ((strcmp (keyword, "SONARTRACEOFF") == 0) ||
             (strcmp (keyword, "SONAR-TRACE-OFF") == 0))
    {
        if (DISPLAYSCREEN) printf ("\n[SONARTRACEOFF] ");
        SONARTRACE = FALSE;
    }
    else if (strcmp (keyword, "PARALLELPORTTRACE") == 0)
    {
        if (DISPLAYSCREEN) printf ("\n[PARALLELPORTTRACE] ");
        PARALLELPORTTRACE = TRUE;
    }
}
#endif
else if ((strcmp (keyword, "TACTICAL") == 0) ||
         (strcmp (keyword, "TACTICAL-HOST") == 0) ||
         (strcmp (keyword, "TACTICALHOST") == 0) ||
         (strcmp (keyword, "STRATEGIC") == 0) ||

```

```

        (strcmp (keyword, "STRATEGICHOST") == 0) ||
        (strcmp (keyword, "STRATEGIC-HOST") == 0))
    {
        if (sscanf (command_buffer, "%s %s", keyword, parameter_string) == 2)
        {
            TACTICAL = TRUE;
            KEYBOARDINPUT = FALSE;
            strcpy (tactical_remote_host_name, parameter_string);
            open_tactical_socket ();
            if (DISPLAYSCREEN)
                printf("\n[%s %s (KEYBOARD-OFF)]",
                    keyword, tactical_remote_host_name);
        }
        else print_help = TRUE;
    }
    else if ((strcmp (keyword, "NO-TACTICAL") == 0) ||
             (strcmp (keyword, "TACTICAL-OFF") == 0))
    {
        if (DISPLAYSCREEN) printf ("\n[%s]\n", keyword);
        TACTICAL = FALSE;
    }
    else if ((strcmp (keyword, "SONARINSTALLED") == 0) ||
             (strcmp (keyword, "SONAR-INSTALLED") == 0))
    {
        if (DISPLAYSCREEN) printf ("\n[SONAR-INSTALLED] ");
        SONARINSTALLED = TRUE;
    }
    else if ((strcmp (keyword, "NOSONARINSTALLED") == 0) ||
             (strcmp (keyword, "NO-SONAR-INSTALLED") == 0))
    {
        if (TRACE && DISPLAYSCREEN) printf ("[NO-SONAR-INSTALLED] ");
        SONARINSTALLED = FALSE;
    }
    else if ((strcmp (keyword, "SONARTRACE") == 0) ||
             (strcmp (keyword, "SONAR-TRACE") == 0) ||
             (strcmp (keyword, "SONAR-TRACE-ON") == 0))
    {
        if (DISPLAYSCREEN) printf ("\n[SONARTRACE] ");
        SONARTRACE = TRUE;
    }
    else if ((strcmp (keyword, "SONARTRACEOFF") == 0) ||
             (strcmp (keyword, "SONAR-TRACE-OFF") == 0))
    {
        if (DISPLAYSCREEN) printf ("\n[SONARTRACEOFF] ");
        SONARTRACE = FALSE;
    }
    else if (strcmp (keyword, "PARALLELPORTTRACE") == 0)
    {
        if (DISPLAYSCREEN) printf ("\n[PARALLELPORTTRACE] ");
        PARALLELPORTTRACE = TRUE;
    }
    else if ((strcmp (keyword, "AUDIBLE") == 0) ||
             (strcmp (keyword, "AUDIO") == 0) ||
             (strcmp (keyword, "AUDIO-ON") == 0) ||
             (strcmp (keyword, "SOUND-ON") == 0) ||
             (strcmp (keyword, "SOUNDON") == 0) ||
             (strcmp (keyword, "SOUND") == 0))
    {
        if (DISPLAYSCREEN) printf ("\n[AUDIBLE] ");
        strcpy (buffer, "AUDIBLE"); /* copy current command to buffer */
        send_buffer_to_virtual_world_socket (); /* send to sound driver */
    }
    else if ((strcmp (keyword, "SILENT") == 0) ||
             (strcmp (keyword, "SILENCE") == 0) ||
             (strcmp (keyword, "NOSOUND") == 0) ||
             (strcmp (keyword, "NO-SOUND") == 0) ||
             (strcmp (keyword, "SOUNDOFF") == 0) ||
             (strcmp (keyword, "SOUND-OFF") == 0) ||

```

```

        (strcmp (keyword, "AUDIOOFF") == 0) ||
        (strcmp (keyword, "AUDIO-OFF") == 0) ||
        (strcmp (keyword, "QUIET") == 0))
    {
        if (DISPLAYSCREEN) printf ("\n[SILENT] ");
        strcpy (buffer, "SILENT"); /* copy current command to buffer */
        send_buffer_to_virtual_world_socket (); /* send to sound driver */
    }
#endif OS9
/* save space due to OS9 size problems */
    else if ((strcmp (keyword, "SOUNDSERIAL") == 0) ||
             (strcmp (keyword, "SOUND-SERIAL") == 0))
    {
        if (TRACE && DISPLAYSCREEN) printf ("\n[SOUNDSERIAL ON] ");
        strcpy (buffer, "SOUNDSERIAL"); /* send precise keyword */
        send_buffer_to_virtual_world_socket (); /* send to sound driver */
        audible_command = FALSE;
    }
    else if ((strcmp (keyword, "SOUNDPARALLEL") == 0) ||
             (strcmp (keyword, "SOUND-PARALLEL") == 0))
    {
        strcpy (buffer, "SOUNDPARALLEL"); /* send precise keyword */
        send_buffer_to_virtual_world_socket (); /* send to sound driver */
        audible_command = FALSE;
    }
    else if ((strcmp (keyword, "EMAIL") == 0) ||
             (strcmp (keyword, "EMAIL-ON") == 0) ||
             (strcmp (keyword, "E-MAIL") == 0) ||
             (strcmp (keyword, "E-MAIL-ON") == 0) ||
             (strcmp (keyword, "EMAILON") == 0))
    {
        if (TRACE && DISPLAYSCREEN) printf ("\n[EMAIL ON] ");
        EMAIL = TRUE;
    }
    else if ((strcmp (keyword, "EMAILOFF") == 0) ||
             (strcmp (keyword, "EMAIL-OFF") == 0) ||
             (strcmp (keyword, "E-MAILOFF") == 0) ||
             (strcmp (keyword, "E-MAIL-OFF") == 0) ||
             (strcmp (keyword, "NO-E-MAIL") == 0) ||
             (strcmp (keyword, "NO-EMAIL") == 0) ||
             (strcmp (keyword, "NO-E-MAIL") == 0) ||
             (strcmp (keyword, "NOEMAIL") == 0))
    {
        if (TRACE && DISPLAYSCREEN) printf ("\n[EMAIL OFF] ");
        EMAIL = FALSE;
    }
#endif
    else if ((strcmp (keyword, "WAYPOINT") == 0) ||
             (strcmp (keyword, "WAYPOINT-ON") == 0))
    {
#ifdef OS9
        parameters_read = sscanf (command_buffer, "%s%lf%lf%lf",
                                   keyword, & parameter1,
                                   & parameter2, & parameter3);
#else
        parameters_read = sscanf (command_buffer, "%s%F%F%F",
                                   keyword, & parameter1,
                                   & parameter2, & parameter3);
#endif
        if (parameters_read == 5)
        {
            if (DISPLAYSCREEN)
                printf ("\n[%s %6.2f %6.2f %6.2f %6.2lf]\n",
                        keyword, parameter1,
                        parameter2, parameter3,
                        parameter4);

            WAYPOINTCONTROL = TRUE;
            FOLLOWWAYPOINTMODE = TRUE;
        }
    }

```

```

        HOVERCONTROL      = FALSE;
        ROTATECONTROL     = FALSE;
        LATERALCONTROL    = FALSE;
        TARGETCONTROL     = FALSE;
        RECOVERYCONTROL   = FALSE;
        INTEGRALDEPTHCONTROL = FALSE;
        time_int_control_on = t + 10.0; /* give PD 10 seconds */
        SONARSCANMODE     = 1;        /* Forward Scan */
        REPORTSTABLE      = TRUE;
        DEADSTICKRUDDER   = FALSE;
        x_command         = parameter1;
        y_command         = parameter2;
        z_command         = parameter3;
        port_rpm_command   = fabs (parameter4); /* ensure fwd */
        stbd_rpm_command   = port_rpm_command; /* motion only */
        DEATH_SPIRAL_RESET = TRUE;
    }
    else if (parameters_read == 4)
    {
        if (DISPLAYSCREEN)
            printf ("\n[%s   %6.2f %6.2f %6.2f]\n",
                    keyword, parameter1,
                    parameter2, parameter3);

        WAYPOINTCONTROL = TRUE;
        FOLLOWWAYPOINTMODE = TRUE;
        HOVERCONTROL     = FALSE;
        ROTATECONTROL    = FALSE;
        LATERALCONTROL   = FALSE;
        TARGETCONTROL    = FALSE;
        RECOVERYCONTROL  = FALSE;
        INTEGRALDEPTHCONTROL = FALSE;
        time_int_control_on = t + 10.0; /* give PD 10 seconds */
        SONARSCANMODE    = 1;        /* Forward Scan */
        REPORTSTABLE     = TRUE;
        DEADSTICKRUDDER  = FALSE;
        x_command        = parameter1;
        y_command        = parameter2;
        z_command        = parameter3;
        port_rpm_command  = fabs (port_rpm_command); /* ensure fwd */
        stbd_rpm_command  = fabs (stbd_rpm_command); /* motion only */
        DEATH_SPIRAL_RESET = TRUE;
    }
    else if (parameters_read == 3)
    {
        if (DISPLAYSCREEN)
            printf ("\n[%s   %6.2f %6.2f]\n", keyword, parameter1,
                    parameter2);

        WAYPOINTCONTROL = TRUE;
        FOLLOWWAYPOINTMODE = TRUE;
        HOVERCONTROL     = FALSE;
        ROTATECONTROL    = FALSE;
        LATERALCONTROL   = FALSE;
        TARGETCONTROL    = FALSE;
        RECOVERYCONTROL  = FALSE;
        INTEGRALDEPTHCONTROL = FALSE;
        time_int_control_on = t + 10.0; /* give PD 10 seconds */
        SONARSCANMODE    = 1;        /* Forward Scan */
        REPORTSTABLE     = TRUE;
        DEADSTICKRUDDER  = FALSE;
        x_command        = parameter1;
        y_command        = parameter2;
        port_rpm_command  = fabs (port_rpm_command); /* ensure fwd */
        stbd_rpm_command  = fabs (stbd_rpm_command); /* motion only */
        DEATH_SPIRAL_RESET = TRUE;
    }
}

```

```

else
{
    WAYPOINTCONTROL = FALSE;
    if (DISPLAYSCREEN)
    {
        printf ("\n warning: improper number of values\n waypoint");
        printf ("set to current position but otherwise ignored\n");
    }
    x_command      = x;
    y_command      = y;
    z_command      = z;
}
if (FOLLOWWAYPOINTMODE)
{
    /* continue until WAYPOINT reached without further script orders */
    time_next_command = t + dt;
    read_another_line = FALSE;
}
}
else if ((strcmp (keyword, "WAYPOINTFOLLOW") == 0) ||
         (strcmp (keyword, "WAYPOINT-FOLLOW") == 0) ||
         (strcmp (keyword, "WAYPOINTFOLLOWON") == 0) ||
         (strcmp (keyword, "WAYPOINT-FOLLOW-ON") == 0))
{
    if (DISPLAYSCREEN) printf ("\n[%s]\n", keyword);
    FOLLOWWAYPOINTMODE = TRUE;
    DEADSTICKRUDDER   = FALSE;
}
else if ((strcmp (keyword, "WAYPOINTFOLLOWOFF") == 0) ||
         (strcmp (keyword, "WAYPOINT-FOLLOW-OFF") == 0))
{
    if (DISPLAYSCREEN) printf ("\n[%s]\n", keyword);
    FOLLOWWAYPOINTMODE = FALSE;
}
else if ((strcmp (keyword, "TARGET-OFF") == 0) ||
         (strcmp (keyword, "TARGETOFF") == 0) ||
         (strcmp (keyword, "NO-TARGET") == 0) ||
         (strcmp (keyword, "NOTARGET") == 0))
{
    TARGETCONTROL      = FALSE;
    RECOVERYCONTROL     = FALSE;
    SONARSCANMODE      = 1;      /* Forward Scan */
    target_bearing_command = 0.0;
    target_range_command  = 0.0;
}
else if ((strcmp (keyword, "TARGET-POINT") == 0) ||
         (strcmp (keyword, "TARGETPOINT") == 0))
{
    TARGETPOINTING = TRUE;
}
else if ((strcmp (keyword, "NO-TARGET-POINT") == 0) ||
         (strcmp (keyword, "NOTARGETPOINT") == 0) ||
         (strcmp (keyword, "TARGET-POINT-OFF") == 0) ||
         (strcmp (keyword, "TARGETPOINTOFF") == 0))
{
    TARGETPOINTING = FALSE;
}
else if ((strcmp (keyword, "STANDOFF") == 0) ||
         (strcmp (keyword, "STAND-OFF") == 0) ||
         (strcmp (keyword, "STANDOFFDISTANCE") == 0) ||
         (strcmp (keyword, "STANDOFF-DISTANCE") == 0) ||
         (strcmp (keyword, "STAND-OFF-DISTANCE") == 0))
{
    parameters_read = sscanf (command_buffer, "%s%lf",
#ifdef os9

```

```

keyword,      & parameter1);
#else
parameters_read = sscanf (command_buffer, "%s%F",
                           keyword,      & parameter1);
#endif

if (parameters_read == 2)
{
    if (DISPLAYSCREEN) printf ("\n[%s   %6.2f]\n",
                               keyword, parameter1);
    standoff_distance = parameter1;

    if (HOVERCONTROL) /* report when stable again */
        REPORTSTABLE = TRUE;
}
else
{
    if (DISPLAYSCREEN)
    {
        printf ("\n[%s]\n", keyword);
        printf (" warning: no standoff value provided, ignored");
    }
}

}
else if ((strcmp (keyword, "HOVEROFF") == 0) ||
         (strcmp (keyword, "HOVER-OFF") == 0) ||
         (strcmp (keyword, "HOVER_OFF") == 0))
{
    if (DISPLAYSCREEN) printf ("\n[HOVER-OFF] ");
    HOVERCONTROL      = FALSE;
    WAYPOINTCONTROL   = FALSE; /*explicitly eliminate side effects */
    FOLLOWWAYPOINTMODE = FALSE;
    port_rpm_command  = 0.0;
    stbd_rpm_command  = 0.0;
    rudder_command    = 0.0;
    read_another_line = FALSE;
}
else if ((strcmp (keyword, "TEXT") == 0) ||
         (strcmp (keyword, "TEXT-ON") == 0))
{
    DISPLAYSCREEN = TRUE;
}
else if ((strcmp (keyword, "NOTEXT") == 0) ||
         (strcmp (keyword, "NO-TEXT") == 0))
{
    DISPLAYSCREEN = FALSE;
}
else if ((strcmp (keyword, "SCANWIDTH") == 0) ||
         (strcmp (keyword, "SCAN-WIDTH") == 0))
{
#ifdef os9
parameters_read = sscanf (command_buffer, "%s%lf",
                           keyword, & parameter1);
#else
parameters_read = sscanf (command_buffer, "%s%F",
                           keyword, & parameter1);
#endif

if (parameters_read == 2)
{
    if ((parameter1 <= 0.0) && DISPLAYSCREEN)
    {
        printf("  illegal SCANWIDTH value, ignored.");
        printf(" [SCANWIDTH = %3.2lf]\n", SCANWIDTH);
    }
    else
    {
        SCANWIDTH      = parameter1;
        if (DISPLAYSCREEN)
            printf("[SCANWIDTH %3.2lf]", SCANWIDTH);
    }
}
}
}

```



```

    }
}
else /* check other possibilities */
{
    parse_mission_string_commands (command_buffer);
}

if (audible_command)
{
    strcpy (buffer, command_buffer); /* copy current command to buffer */
    send_buffer_to_virtual_world_socket (); /* send to sound driver */
}

if ((print_help) && DISPLAYSCREEN)
{
    if (DISPLAYSCREEN) printf ("%s", command_buffer);
    print_valid_keywords ();

    strcpy (buffer, " is an unknown command"); /* copy msg to buffer */
    send_buffer_to_virtual_world_socket (); /* send to sound driver */
}

return_value = print_help;
print_help = FALSE; /* reset value */

if (TACTICAL)
{
    time_next_command = t + dt; /* one command per timestep only */
    read_another_line = FALSE; /* force acknowledgement and loop*/
    /* TIME and WAIT and RUN commands are not needed in TACTICAL mode */
}
if ((HOVERCONTROL) || (WAYPOINTCONTROL) || (REPORTSTABLE))
{
    read_another_line = FALSE; /* force acknowledgement and loop*/
    /* TIME and WAIT and RUN commands are not needed in TACTICAL mode */
}
} /* loop until read_another_line is FALSE) */

if (TRACE && DISPLAYSCREEN)
    printf ("\n[end parse_mission_script_commands ()]\n");

return (return_value);

} /* end parse_mission_script_commands () */

/*****
void parse_mission_string_commands (command)
char * command;
{
    int    index;
    int    number_values = 0;
    char   parameter_string [60];

    if (NOSCRIPT) return; /* no script, telemetry playback only */

    if (TRACE && DISPLAYSCREEN)
        printf ("\n[parse_mission_string_commands start]\n");

    number_values = sscanf (command_buffer, "%s", keyword);

    for (index = 0; index <= (int) strlen (keyword); index++)
        keyword [index] = toupper (keyword [index]);

    if    (number_values != 1)
    {

```

```

        if (DISPLAYSCREEN) printf (" [no parse word found]\n");
        return;
    }
    if ((strcmp (keyword, "VIRTUALHOST") == 0) ||
        (strcmp (keyword, "VIRTUAL") == 0) ||
        (strcmp (keyword, "VIRTUAL-HOST") == 0) ||
        (strcmp (keyword, "REMOTE") == 0) ||
        (strcmp (keyword, "REMOTEHOST") == 0) ||
        (strcmp (keyword, "REMOTE-HOST") == 0) ||
        (strcmp (keyword, "DYNAMICS") == 0))
    {
        if (sscanf (command, "%s %s", keyword, parameter_string) == 2)
        {
            strcpy (virtual_world_remote_host_name, parameter_string);
            if (DISPLAYSCREEN) printf ("\n[VIRTUAL-HOST %s] ",
                                     virtual_world_remote_host_name);
        }
        else print_help = TRUE;
    }
    else print_help = TRUE; /* invalid command line entry parameter found */

    if (TRACE && DISPLAYSCREEN)
        printf ("\n[parse_mission_string_commands complete]\n");

    return;

} /* end parse_mission_string_commands () */

/*****

void print_valid_keywords ()
{
    if (TACTICAL || TACTICALPARSE) return;

    printf ("\n");
    printf ("          [help] [trace|notrace] [loopforever|looponce]\n");
    printf ("          [wait #] [time #] [timestep (0.0..5.0)] [mission]\n");
    printf ("          [keyboard|keyboard-off] [quit] [kill]\n");
    printf ("          [rpm] [course] [depth] [thrusters|thrusters-off]\n");
    printf ("          [loopfilebackup] [entercontrolconstants]\n");
    printf ("          [rotate] [position|location|fix] [orientation]\n");
    printf ("          [gps|gps-fix] [gps-complete|gps-fix-complete] \n");
    printf ("          [sonartrace|sonartraceoff] [sonarinstalled]\n");
    printf ("          [trace|trace-off] [parallelexporttrace] \n");
    printf ("          [remotehost hostname] [realtime|nopause] [pause]\n");
    printf ("          [loop-forever|loop-once] [controlconstantsfile]\n");
    printf ("          [silence] [e-mail|no-email] [waypoint]\n\n");
    printf ("See ~/execution/mission.script.HELP for command syntax details.\n");
    printf ("\n");

#ifdef sgi
    /* don't pop up help file if TACTICAL is running or invoking this code */
    if ((HELPPFILELAUNCHED == FALSE) && (TACTICAL == FALSE) &&
        (TACTICALPARSE == FALSE))
    {
        printf ("popping up 'mission.script.HELP' as a zip file...\n");
        system ("zip -v ~/execution/mission.script.HELP");
        HELPPFILELAUNCHED = TRUE;
    }
#endif

    return;

} /* end print_valid_keywords */

/*****/

```

```

void get_control_constants ()
/* get data from file at program start */
{
    if (TACTICALPARSE) return;

    if (TRACE && DISPLAYSCREEN)
        printf ("\n[start get_control_constants ()]\n");

    if (ENTERCONTROLCONSTANTS && DISPLAYSCREEN) /* - - - - - */
    {
        printf ("Error: ENTERCONTROLCONSTANTS not possible while ");
        printf ("DISPLAYSCREEN is FALSE.\n");
        printf ("Using originally loaded control constants.\n");
        ENTERCONTROLCONSTANTS = FALSE;
    }
    else if (ENTERCONTROLCONSTANTS) /* - - - - - */
    {
        printf("Input start_dwell (startup delay time in seconds)\n");
        scanf("%d", &start_dwell);

        /* note %F required by OS-9, accepted by SGI as equivalent to %lf */
        printf("Input k_psi, k_r, k_v\n");
#ifdef os9
        scanf("%lf %lf %lf", &k_psi, &k_r, &k_v);
#else
        scanf("%F %F %F", &k_psi, &k_r, &k_v);
#endif

        printf("Input k_z, k_w, k_theta, and k_q\n");
#ifdef os9
        scanf("%lf %lf %lf %lf", &k_z, &k_w, &k_theta, &k_q);
#else
        scanf("%F %F %F %F", &k_z, &k_w, &k_theta, &k_q);
#endif

        printf("Input k_thruster_psi, k_thruster_r\n");
#ifdef os9
        scanf("%lf %lf", &k_thruster_psi, &k_thruster_r);
#else
        scanf("%F %F", &k_thruster_psi, &k_thruster_r);
#endif

        printf("Input k_thruster_rotate\n");
#ifdef os9
        scanf("%lf", &k_thruster_rotate);
#else
        scanf("%F", &k_thruster_rotate);
#endif

        printf("Input k_thruster_z, k_thruster_w\n");
#ifdef os9
        scanf("%lf %lf", &k_thruster_z, &k_thruster_w);
#else
        scanf("%F %F", &k_thruster_z, &k_thruster_w);
#endif

        printf("Input k_thruster_theta\n");
#ifdef os9
        scanf("%lf", &k_thruster_theta);
#else
        scanf("%F", &k_thruster_theta);
#endif

        printf("Input k_propeller_hover, k_surge_hover, k_propeller_current\n");
#ifdef os9
        scanf("%lf %lf %lf", &k_propeller_hover, &k_surge_hover,
#else

```

```

scanf("%F %F %F",    &k_propeller_hover, &k_surge_hover,
#endif
                                &k_propeller_current);

printf("Input k_thruster_hover, k_sway_hover, k_thruster_current\n");
#endif os9
scanf("%lf %lf %lf", &k_thruster_hover,
#else
scanf("%F %F %F",    &k_thruster_hover,
#endif
                                &k_sway_hover, &k_thruster_current);

printf("Input k_thruster_lateral\n");
#endif os9
scanf("%lf", &k_thruster_lateral);
#else
scanf("%F", &k_thruster_lateral);
#endif
}
else if (LOADCONTROLCONSTANTS) /* - - - - - */
{
if ((controlconstantsinputfile = fopen (CONTROLCONSTANTSINPUT-
NAME,"r"))
    == NULL)
{
printf("AUV execution: unable to open control constants input file ");
printf("%s for reading.\n", CONTROLCONSTANTSINPUTNAME);
printf
("          Check ownership permissions in current directory.\n");
printf("Exit.\n");
exit (-1);
}

strcpy (buffer, "Control constants file is");

if (TRACE && DISPLAYSCREEN)
{
printf ("\n[controlconstantsinputfile %s open, pointer = %x]\n",
CONTROLCONSTANTSINPUTNAME, controlconstantsinputfile);
send_buffer_to_virtual_world_socket (); /* buffer message sent */
}

strcpy (buffer, CONTROLCONSTANTSINPUTNAME);
if (TRACE && DISPLAYSCREEN)
{
send_buffer_to_virtual_world_socket (); /* buffer message sent */
}

start_dwell = 1; /* delay time in seconds */
/* skip remaining header lines in file */
for (i=1;i<=8;i++) fgets (local_buffer, 80, controlconstantsinputfile);

/* note %F required by OS-9, accepted by SGI as equivalent to %lf */
#endif os9
fscanf (controlconstantsinputfile, "%lf %lf %lf", &k_psi, &k_r, &k_v);
#else
fscanf (controlconstantsinputfile, "%F %F %F",    &k_psi, &k_r, &k_v);
#endif

#endif os9
fscanf (controlconstantsinputfile, "%lf %lf %lf %lf", &k_z, &k_w,
#else
fscanf (controlconstantsinputfile, "%F %F %F %F",    &k_z, &k_w,
#endif
                                &k_theta, &k_q);

for (i=1;i<=5;i++) fgets (local_buffer, 80, controlconstantsinputfile);
#endif os9

```

```

        fscanf(controlconstantsinputfile, "%lf %lf", &k_thruster_psi,
#else
        fscanf(controlconstantsinputfile, "%F %F", &k_thruster_psi,
#endif
                                &k_thruster_r);

#ifdef os9
        fscanf(controlconstantsinputfile, "%lf", &k_thruster_rotate);
#else
        fscanf(controlconstantsinputfile, "%F", &k_thruster_rotate);
#endif

        for (i=1;i<=5;i++) fgets (local_buffer, 80, controlconstantsinputfile);
#ifdef os9
        fscanf(controlconstantsinputfile, "%lf %lf", &k_thruster_z,
#else
        fscanf(controlconstantsinputfile, "%F %F", &k_thruster_z,
#endif
                                &k_thruster_w);
#ifdef os9
        fscanf(controlconstantsinputfile, "%lf", &k_thruster_theta);
#else
        fscanf(controlconstantsinputfile, "%F", &k_thruster_theta);
#endif

        for (i=1;i<=5;i++) fgets (local_buffer, 80, controlconstantsinputfile);
#ifdef os9
        fscanf(controlconstantsinputfile, "%lf %lf %lf", &k_propeller_hover,
#else
        fscanf(controlconstantsinputfile, "%F %F %F", &k_propeller_hover,
#endif
                                &k_surge_hover,
                                &k_propeller_current);

        for (i=1;i<=5;i++) fgets (local_buffer, 80, controlconstantsinputfile);
#ifdef os9
        fscanf(controlconstantsinputfile, "%lf %lf %lf", &k_thruster_hover,
#else
        fscanf(controlconstantsinputfile, "%F %F %F", &k_thruster_hover,
#endif
                                &k_sway_hover,
                                &k_thruster_current);

        for (i=1;i<=5;i++) fgets (local_buffer, 80, controlconstantsinputfile);
#ifdef os9
        fscanf(controlconstantsinputfile, "%lf", &k_thruster_lateral);
#else
        fscanf(controlconstantsinputfile, "%F", &k_thruster_lateral);
#endif
    }
    else /* use default initialization values - - - - - */
    {
        if (TRACE && DISPLAYSCREEN)
            printf ("\n[using default control constant values]\n");

        start_dwell = 1; /* delay time in seconds */

        k_psi = 1.00; /* degrees rudder per degree of course error */
        k_r = 2.00; /* degrees rudder per degree/sec yaw rate */
        k_v = 0.00; /* needed ?? */

        k_z = 15.0; /* degrees planes per foot of depth error */
        k_w = 2.0;
        k_theta = 4.0;
        k_q = 1.0;

        rpm = 400.0;
    }

```

```

    k_thruster_psi    = 0.6; /* volts per 1 degree course error      */
    k_thruster_r      = 5.0;
    k_thruster_rotate = 1.5; /* (24V)^2 => 2 # = 16.0 deg/sec empirical */
                          /* k_thruster_rotate = (24V / 16 deg/sec)^2 */
    k_thruster_z      = 20.0; /* guess 20 fresh water, 30 in sea water */
    k_thruster_w      = 80.0;
    k_thruster_theta  = 1.0;

    k_propeller_hover = 200.0; /* 200 rpm per one foot error      */
    k_surge_hover     = 6000.0; /* 60 rpm per 0.01 foot/sec surge    */
                          /* this value is high to reduce sternway */
    k_propeller_current = 6500.0; /* experimental */

    k_thruster_hover   = 4.0;
    k_sway_hover       = 40.0;
    k_thruster_current = 40.0; /* experimental */

    k_thruster_lateral = 48.0; /* 24 V = 2 # = 0.5 ft/sec empirically */
                          /* note voltage follows a square law      */
}

if (DISPLAYSCREEN &&
    (ENTERCONTROLCONSTANTS || LOADCONTROLCONSTANTS || SHOWCONTROLCONSTANTS))
{
    printf ("\n");
    printf ("n[k_psi = %5.2f, k_r = %5.2f, k_v = %5.2f, k_z = %5.2f, ",
            k_psi, k_r, k_v, k_z);
    printf ("k_w = %5.2f, k_theta = %5.2f, k_q = %5.2f]\n",
            k_w, k_theta, k_q);
    printf ("n[k_thruster_psi = %5.2f, k_thruster_r = %5.2f, ",
            k_thruster_psi, k_thruster_r);
    printf ("k_thruster_rotate = %5.2f]\n",
            k_thruster_rotate);

    printf (" [k_thruster_z = %5.2f, k_thruster_w = %5.2f, ",
            k_thruster_z, k_thruster_w);
    printf ("k_thruster_theta = %5.2f]\n",
            k_thruster_theta);

    printf ("n[k_propeller_hover = %5.2f, k_surge_hover = %5.2f]\n",
            k_propeller_hover, k_surge_hover);

    printf ("n[k_propeller_current = %5.2f]\n",
            k_propeller_current);

    printf ("n[k_thruster_hover = %5.2f, k_sway_hover = %5.2f]\n",
            k_thruster_hover, k_sway_hover);

    printf ("n[k_thruster_current = %5.2f]\n",
            k_thruster_current);

    printf ("n[k_thruster_lateral = %5.2f]\n\n", k_thruster_lateral);
}

if (SHOWCONTROLCONSTANTS == TRUE)
{
    SHOWCONTROLCONSTANTS = FALSE;
    if (TRACE && DISPLAYSCREEN)
        printf ("n[finish get_control_constants ()]\n");
    return;
}

if ((controlconstantsoutputfile = fopen (CONTROLCONSTANTSOUTPUTNAME, "w"))
    == NULL)
{
    printf("AUV execution: unable to open control constants output file ");
    printf("%s for writing.\n", CONTROLCONSTANTSOUTPUTNAME);
    printf

```

```

    ("          Check ownership permissions in current directory.\n");
    printf("Exit.\n");
    exit (-1);
}
if (TRACE && DISPLAYSCREEN)
    printf ("\n[controlconstantsoutputfile %s open, pointer = %x]\n",
            CONTROLCONSTANTSOUTPUTNAME, controlconstantsoutputfile);

/* warning: the file read capability depends on file format/line spacing */
fprintf (controlconstantsoutputfile,
        " _____ \n\n");
fprintf (controlconstantsoutputfile,
        " AUV execution level control algorithm coefficients \n");
fprintf (controlconstantsoutputfile,
        " _____ \n\n\n");
fprintf (controlconstantsoutputfile,
        " k_psi k_r k_v k_z k_w k_theta k_q\n\n");
fprintf (controlconstantsoutputfile,
        " %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f %5.2f\n\n\n\n",
        k_psi, k_r, k_v, k_z, k_w, k_theta, k_q);

fprintf (controlconstantsoutputfile,
        " k_thruster_psi k_thruster_r k_thruster_rotate\n\n");
fprintf (controlconstantsoutputfile,
        " %5.2f %5.2f %5.2f\n\n\n\n",
        k_thruster_psi, k_thruster_r, k_thruster_rotate);

fprintf (controlconstantsoutputfile,
        " k_thruster_z k_thruster_w k_thruster_theta\n\n");
fprintf (controlconstantsoutputfile,
        " %5.2f %5.2f %5.2f \n\n\n\n",
        k_thruster_z, k_thruster_w, k_thruster_theta);

fprintf (controlconstantsoutputfile,
        " k_propeller_hover k_surge_hover k_propeller_current\n\n");
fprintf (controlconstantsoutputfile,
        " %5.2f %5.2f %5.2f\n\n\n\n",
        k_propeller_hover, k_surge_hover, k_propeller_current);

fprintf (controlconstantsoutputfile,
        " k_thruster_hover k_sway_hover k_thruster_current\n\n");
fprintf (controlconstantsoutputfile,
        " %5.2f %5.2f %5.2f\n\n\n\n",
        k_thruster_hover, k_sway_hover, k_thruster_current);

fprintf (controlconstantsoutputfile,
        " k_thruster_lateral \n\n");
fprintf (controlconstantsoutputfile,
        " %5.2f\n\n\n\n",
        k_thruster_lateral);

fflush (controlconstantsoutputfile); /* force completion of file write */
fclose (controlconstantsoutputfile);

if (TRACE && DISPLAYSCREEN)
    printf ("\n[finish get_control_constants ()]\n");

return;
} /* end get_control_constants () */
/*****
/* end parse_functions.c */
*****/

```

```

/*****
/*
Program:      execution.c

Description:   AUV execution level program

Authors:      Don Brutzman, Mike Burns, Duane Davis,
              Dave Marco & Walt Landaker

Revised:      2 August 96

System:       AUV Gespac 68020/68030, OS-9 version 2.4
Compiler:     Gespac cc Kernighan & Richie (K&R) C (NOT ANSI C!)

Compilation:  ftp>      put execution.c
              auvsim1> chd execution
              [68020] auvsim1> make -k2f execution
              [68030] auvsim1> make      execution

              [Irix ] ~brutzman/execution>> make execution

Execution:
  [Irix ] ~brutzman/execution>> cd execution
          ~brutzman/execution>> execution remote dynamics-hostname

          where dynamics-hostname is the IP name of the host running
          the dynamics (virtual world) program

Plotting:     see gnuplot scripts 'auv_plot.gnu' & 'auv_plot_1_second.gnu'
          ~brutzman/execution>> gnuplot auv_plot.gnu

Debugging:    ~brutzman/execution>> lint -lm execution.c

              lint -lm -Iglobals.h -Idefines.h globals.c parse_functions.c \
                  execution.c

              ~brutzman/execution>> make warnings

Description:   closed loop for operation during vehicle in-water
              missions as well as in virtual world

Active changes: Don Brutzman    working lab/virtual world networked version
              & sonar tracking and target station control

Future work:   Sonar/altimeter integration code reintegrated/retested

              Audios seem to be generated differently by OS-9

              standardize parsing of command line and script commands

              fix pitch control

              finish sliding mode control

Testing interprocessor connections:

parallel port  /P      OS-9 auvsim1> mfi_a3

              LPT1:    DOS auvsim2> portfix
                      > print filename.txt

serial port   (/T1) /TT OS-9 auvsim1> wr2t1          then write text
                  OS-9 auvsim1> rdt1a          then read text
                  DOS auvsim2> C:\COMM\PROCOMM
                      then <alto> for chat mode
                      <altF10> help, <altX> exit

```


Interfaces:

Telemetry sent via serial port /tt [== /t1 at high baud rate]
Telemetry received via parallel port /P

Telemetry is optionally passed to/from tactical level running on 80386

Reads files: mission.init [mission initialization data file]
Writes files: output.data [vehicle telemetry state vector data]
 output.auv [tactical order/executive report log]

Sonar commands/replies via device port /t3

Note that %F double formats are used instead of %lf on scanf() and sscanf()
calls for OS-9 compatibility. SGI C compiler does not complain.
gcc on Sun does fail at run-time, so ifdef's are used to support
the proper format string.

```

*/
/*****/

#include "globals.h"
#include "statevector.h"
#include "defines.h"

/*****/
/* function prototypes */
/* is there some way to put parameter specifications in the prototypes??
*/
/*    only if we buy the ANSI C compiler from Microware (or shift to VxWorks)
*/

/* thus following prototypes are missing parameters :( */

void            closed_loop_control_module            ();

double          read_depth                            ();
double          read_psi                             ();
double          read_roll_rate_gyro                  ();
double          read_pitch_rate_gyro                 ();
double          read_yaw_rate_gyro                   ();
double          read_port_motor_rpm                  ();
double          read_stbd_motor_rpm                  ();
double          read_motor                           ();
double          read_roll_angle                      ();
double          read_pitch_angle                     ();
double          read_heading                         ();
double          read_speed                           ();

void            kalman_z                             ();
void            kalman_sonar1000                     ();
void            kalman_sonar725                      ();

void            XY_psi_model_est                     ();

double          read_computer_battery_voltage       ();
double          read_motor_gyro_battery_voltage     ();
void            XY_model_est                         ();
int             leak_check                           ();

void            zero_gyro_data                       ();
void            zero_surfaces                        ();
void            initialize_adcs                      ();
void            init_pia                             ();
void            init_tim1a                           ();
void            thruster_power                       ();

```

```

void        screw_power                ();
void        command_control_surface    ();
void        command_rudder             ();
void        command_planes             ();
void        command_propellors_off     ();
void        command_thrusters_off      ();
void        command_motor              ();
void        test_alive                 ();

void        get_init_avg               ();
void        get_avg_rng               ();

void        open_device_paths          ();
void        tty_mode                   ();
void        close_device_paths         ();
void        read_parallel_port         ();

unsigned char read_tim1ac1             ();
void        write_tim1a               ();

void        send_dac1                 ();
void        send_dac2b                ();
int         get_adc1                  ();
int         get_adc2                  ();

void        Init_PortA                ();
void        Init_PortB                ();
unsigned char Read_PortA              ();
unsigned char Read_PortB              ();
unsigned short Read_PortAB            ();
void        set_bsyA                  ();
void        rst_bsyA                  ();
int         ck_sta                    ();

void        initialize_sonar          ();
char        set_scanning_gain         ();
char        send_sonar_command        ();
void        center_sonar              ();
void        set_step_size              ();
void        step_sonar                ();
void        ping_sonar                ();
double      read_sonar                ();
void        control_sonar             ();

void        open_virtual_world_socket ();
void        shutdown_virtual_world_socket ();
void        send_buffer_to_virtual_world_socket ();
void        get_string_from_virtual_world_socket ();

void        record_data               ();

void        execute_shutdown_script   ();

void        compute_hover_controls    ();
void        compute_target_controls   ();
void        compute_recovery_controls ();
void        compute_waypoint_controls ();
void        compute_lateral_controls ();
void        compute_rotate_controls   ();
void        compute_lateral_thrusters ();
void        compute_vertical_thrusters ();
void        compute_fin_controls      ();

```

/* Functions added to implement Dave M's speed control */

```

int         port_speed_control        ();
int         stbd_speed_control        ();

```

```

/* Dive Tracker Functions */

int      createdmod      ();
int      CLReaddmod      ();
void *   AttachMod       ();
int      DettachMod      ();
void *   CreateMod       ();

/*****
/* external function prototypes */
/* from external_functions.c */

extern double      degrees      ();
extern double      radians      ();
extern double      normalize    ();
extern double      normalize2   ();
extern double      radian_normalize    ();
extern double      radian_normalize2   ();
extern void        clamp       ();

extern double      atan2        ();
extern double      atan2z       ();
extern double      sinh         ();
extern double      cosh         ();
extern double      tanh         ();

extern void        build_telemetry_string      ();
extern void        parse_telemetry_string      ();
extern void        read_telemetry_string       ();

extern void        open_tactical_socket        ();
extern void        shutdown_tactical_socket    ();
extern void        send_buffer_to_tactical_socket ();
extern void        get_string_from_tactical_socket ();

extern void        record_data_on              ();
extern void        record_data_off             ();

extern void        cage_dg                    ();
extern void        uncage_dg                  ();

extern int         detect_death_spiral         ();

/* from parse_functions.c */

extern void        parse_command_line_flags    ();
extern int         parse_mission_script_commands ();
extern void        parse_mission_string_commands ();

extern void        print_valid_keywords        ();

extern void        get_control_constants       ();

extern double      dsign                    ();
extern double      dtanh                    ();

*****/

/* File Scope Globals for use by the Thruster Speed Controller Routines */

double Int_rs = 0.0,
       Int_ls = 0.0;

/* DAC Values being sent 50 props and thrusters */

```

```

int    v_dls = 512,
       v_drs = 512,
       v_dblt = 512,
       v_dslt = 512,
       v_dbvt = 512,
       v_dsvt = 512;

double sin_psi,
       cos_psi,
       cos_phi;

/* Dive Tracker Variables */

#ifdef os9
DT2CLMem  *dt_dmod;
#endif

/* File Scope Globals for Target Control Routines */

int    new_target_update      = 0;
int    st1000_bytes_expected  = 0;
double psi_command_tgt        = 0.0;
double time_last_target_update = 0.0;

int    SONARPINGED            = *FALSE;
int    reset_sonar_filter      = TRUE;
double ST1000_range_kal;
double ST1000_range_kal_dot;
double ST1000_range_kal_ddot;
double ST725_range_kal;
double ST725_range_kal_dot;
double ST725_range_kal_ddot;

/*****
/*****
main (argc, argv)          /* Note K+R C function prototyping is due to OS-9. */
    int argc; char **argv; /* command line arguments. */
{
    if (TRACE && DISPLAYSCREEN) printf ("[start main:  execution]\n");
    strcpy (virtual_world_remote_host_name, VIRTUAL_WORLD_REMOTE_HOST_NAME);
    strcpy (tactical_remote_host_name, TACTICAL_REMOTE_HOST_NAME);
    dt = TIMESTEP;
    parse_command_line_flags (argc, argv);
    open_device_paths ();
    kal_init_z = TRUE;
    record_data_on (); /* Open files for data logging */
    if (LOCATIONLAB)
    {
        open_virtual_world_socket (); /* open connection to virtual world */
        if (TRACE && DISPLAYSCREEN)
            printf ("\n[LOCATIONLAB == TRUE, open_virtual_world_socket ()]\n");
        if (strlen (buffer) > 0)
            send_buffer_to_virtual_world_socket ();
            /* SILENT? send to sound driver */
        strcpy (buffer, " A U V virtual world socket is open");
    }
}

```

```

        send_buffer_to_virtual_world_socket ();
        /* buffer containing message sent */
    }

    if (TACTICAL)
    {
        open_tactical_socket ();          /* open connection to tactical level */

        strcpy (buffer, " A U V tactical socket is open");
        if (DISPLAYSCREEN) printf ("%s\n", buffer);
        send_buffer_to_virtual_world_socket (); /*buffer containing message sent*/
    }
    if (LOCATIONLAB == FALSE)
    {
#ifdef os9
        /* Dive Tracker Stuff */
        if (DIVETRACKER)
        {
            createdmod();

            /* Fork Dive Tracker Process */
            if ((dt_pid =
                os9fork("/r0/div_trac",0,dt_fork_parmptr,0,0,0)) > 0)
            {
                if (DISPLAYSCREEN)
                    printf("[Dive Tracker Process %d forked]\n",dt_pid);
            }
            else
            {
                if (DISPLAYSCREEN) printf("[Can't Fork Dive Tracker Process]\n");
            }
        }
    }
#endif

    /* sleep gives 5 minutes to unhook wires and put AUV in the water */
    if (DISPLAYSCREEN == FALSE)
    {
        printf("[Starting a 60 second sleep, disconnect Ethernet cable!]\n");
        sleep (60); /* disconnect cable during this interval */
    }
    init_pia ();
    init_tim1a (1);
    uncage_dg ();

    zero_surfaces ();
    command_propellors_off ();
    command_thrusters_off ();
    thruster_power(1);
    screw_power(1);
    initialize_adcs();
    zero_gyro_data ();
}

if (SONARINSTALLED)
{
    if (LOCATIONLAB == FALSE)
    {
        initialize_sonar ();
        set_step_size (1); /* set step size to 0.9 */
    }
    center_sonar (); /* must have open_device_paths 1st */
}

get_control_constants ();          /* announce filename as diagnostic */
strcpy (buffer, "EXECUTION_INITIALIZED");

```

```

send_buffer_to_tactical_socket (); /* message */
tsleep(10);

z = read_depth (); /* Read depth */
z_command = z;

if (LOCATIONLAB)/*pass initial position to initialize hydrodynamics model*/
{
    sprintf (buffer, "initial position %4.1lf %4.1lf %4.1lf ", x, y, z);
    if (DISPLAYSCREEN) printf ("%s\n", buffer);
    send_buffer_to_virtual_world_socket ();/*buffer containing message sent*/

    sprintf (buffer, "position %4.1lf %4.1lf %4.1lf ", x, y, z); /* silent */
    send_buffer_to_virtual_world_socket ();/*buffer containing message sent*/
}

EMAIL_ENTERED = FALSE;

/
*****
do /* while (LOCATIONLAB && LOOPFOREVER) */
{ /* indefinite repeat loop for long-duration lab testing*/
/*-----*/

if (DISPLAYSCREEN)
{
    if (LOCATIONLAB && (EMAIL) && (EMAIL_ENTERED == FALSE))
    {
        strcpy (buffer, " Please Enter Your E-mail Address");
        send_buffer_to_virtual_world_socket ();/*buffer containing msg sent */
        if (DISPLAYSCREEN) printf ("%s *** HERE ***: ", buffer);
        strcpy (email_address, "");
        gets (email_address);
        EMAIL_ENTERED = TRUE;
        sprintf (buffer, "Thanks");
        if (DISPLAYSCREEN) printf ("%s ", buffer);
        send_buffer_to_virtual_world_socket ();/*buffer containing msg sent */

        if ((int) (strlen (email_address) > 2))
        {
            sprintf (buffer, "%s\n", email_address);
            if (DISPLAYSCREEN) printf ("%s\n", buffer);
            send_buffer_to_virtual_world_socket (); /* buffer sent */

            if ((strcmp (email_address, "brutzman") != 0) &&
                (strcmp (email_address, "BRUTZMAN") != 0) &&
                (strcmp (email_address, "brutzman@nps.navy.mil") != 0))
            {
                emailaddressfile = fopen(EMAILADDRESSFILENAME,"a"); /*append */
                fprintf (emailaddressfile, "%s\n", email_address);
                fclose (emailaddressfile);
            }
        }
    }
    else if (LOCATIONLAB == FALSE) /* in water */
    {
        test_alive (10, start_dwell);
    }
    if (DISPLAYSCREEN) printf("\n\nOK!! Starting the mission.\n");
}

if (NOSCRIPT == FALSE) /* scriptfile available */
{
    parse_mission_script_commands (); /* read initial script orders */
    /* ignore failure */
}

strcpy (buffer, " , , , ,"); /* pause */

```

```

send_buffer_to_virtual_world_socket (); /* buffer containing msg sent */
strcpy (buffer, " A U V is starting");
send_buffer_to_virtual_world_socket (); /* buffer containing msg sent */

/* Initialization of closed loop parameters */

buffer_index      = 0;
telemetry_records_saved = 0;
mission_leg_counter = 0;
end_test          = FALSE;
wrap_count        = 0;
t                 = 0.0;
dt_time           = 0.0;

/*-----*/
/* Main program operational loop code */
if (TRACE && DISPLAYSCREEN)
    printf ("[Starting main program operational loop code... ]");

nextloopclock = clock () + (int)(dt * (double) CLOCKS_PER_SEC);

/*-----*/
while (end_test == FALSE) /*this is the realtime main operational loop */
    /* when end_test == TRUE then loop is done */
    {
        closed_loop_control_module (); /* closed loop code is here <----- */
    }
    /* end of real-time main operational loop */

/* Kill Dive Tracker Process, Unlink Shared Memory, and Cage Gyro */
if (LOCATIONLAB == FALSE)
{
    cage_dg();
#ifdef os9
    /* Kill Divetracker Process */
    if (DIVETRACKER)
    {
        kill(dt_pid,0);

        /* Unlink Shared Memory Module */
        ul_pid = os9exec(os9forkc,argblk[0],argblk,enviro,0,0,3);
        ul_pid = os9exec(os9forkc,argblk[0],argblk,enviro,0,0,3);
        ul_pid = os9exec(os9forkc,argblk[0],argblk,enviro,0,0,3);
    }
#endif
}

/*-----*/
/* lab version may repeat forever for long-duration testing: */
replication_count ++;

if (LOCATIONLAB && LOOPFOREVER && DISPLAYSCREEN)
{
    printf ("\n[LOOP FOREVER enabled, next loop is replication %d...]\n",
            replication_count);
    sprintf (buffer, " LOOP FOREVER enabled, next loop is replication");
    send_buffer_to_virtual_world_socket (); /* buffer msg sent */
    sprintf (buffer, " %d", replication_count);
    send_buffer_to_virtual_world_socket (); /* buffer msg sent */
}

```

```

if (LOCATIONLAB && LOOPFOREVER)
{
    /* reset amount of time to wait for next command */
    time_next_command = 0.0;
    t = 0.0;
    if (DISPLAYSCREEN)
    {
        printf ("\nLoopforever reset time: [time_next_command = 0.0] ");
        printf (" [t = 0.0]\n");
    }
}

if (LOOPFILEBACKUP)
{
    record_data_off ();

#ifdef os9
    if (DISPLAYSCREEN)
        printf ("rm                                output.telemetry.previous\n");
    system ("rm                                output.telemetry.previous" );
    if (DISPLAYSCREEN)
        printf ("cp mission.output.telemetry output.telemetry.previous\n");
    system ("cp mission.output.telemetry output.telemetry.previous" );
    if (DISPLAYSCREEN) printf ("rm                                out-
put.1_second.previous\n" );
    system ("rm                                output.1_second.previous" );
    if (DISPLAYSCREEN)
        printf ("cp mission.output.1_second output.1_second.previous\n" );
    system ("cp mission.output.1_second output.1_second.previous" );
#else
    if (DISPLAYSCREEN)
        printf ("del                                output.telemetry.previous\n");
    system ("del                                output.telemetry.previous" );
    if (DISPLAYSCREEN)
        printf ("copy mission.output.telemetry output.telemetry.previous\n");
    system ("copy mission.output.telemetry output.telemetry.previous" );
    if (DISPLAYSCREEN)
        printf ("del                                output.1_second.previous\n" );
    system ("del                                output.1_second.previous" );
    if (DISPLAYSCREEN)
        printf ("copy mission.output.1_second output.1_second.previous\n" );
    system ("copy mission.output.1_second output.1_second.previous" );
#endif

    if (LOCATIONLAB)
    {
        strcpy (buffer, " telemetry data backup complete");
        send_buffer_to_virtual_world_socket (); /* buffer msg sent */
    }
}
else /* don't bother backing up most recent results */
{
    if (LOCATIONLAB && LOOPFOREVER)
    {
        if (auvtextfile) rewind (auvtextfile);
        if (((TACTICAL == FALSE) || (TACTICALPARSE))
            && (auvdatafile != NULL))
        {
            rewind (auvdatafile);
            if (TRACE && DISPLAYSCREEN)
                printf ("[auvtextfile & auvdatafile rewound to ");
            printf ("output.data.previous & output.auv.previous]\n");
        }
        strcpy (buffer, " telemetry data backup skipped");
        send_buffer_to_virtual_world_socket (); /* buffer msg sent */
    }
}

```



```

    }

    if (NOSCRIPT == FALSE)
    {
        if (auvscriptfile)
        {
            fflush (auvscriptfile); /* force completion-file write */
            if (fclose (auvscriptfile) == 0)
            {
                if (DISPLAYSCREEN)
                    printf ("[success closing auvscriptfile mission.script.backup]\n");
            }
            else if (DISPLAYSCREEN)
                printf ("[failure closing auvscriptfile mission.script.backup]\n");
        }
        /* - - - - - orders - - - - - */

        if (auvordersfile) fflush(auvordersfile); /*force write completion*/
        if (auvordersfile) fclose(auvordersfile);

/*
#ifdef os9
    if (DISPLAYSCREEN)
        printf ("rm                                mission.output.orders\n");
    system ("rm                                mission.output.orders" );
    if (DISPLAYSCREEN)
        printf ("mv mission.output.orders.backup mission.output.orders\n");
    system ("mv mission.output.orders.backup mission.output.orders" );
#else
    if (DISPLAYSCREEN)
        printf ("del                                mission.output.orders\n");
    system ("del                                mission.output.orders" );
    if (DISPLAYSCREEN)
        printf ("copy mission.output.orders.backup mission.output.orders\n");
    system ("copy mission.output.orders.backup mission.output.orders" );
    if (DISPLAYSCREEN) printf ("del mission.output.orders.backup\n\n");
    system ("del mission.output.orders.backup" );
#endif
*/

#ifdef os9
    sprintf (buffer, "rm %s.backup\n",                AUVORDERSFILENAME);
#else
    sprintf (buffer, "del %s.backup\n",                AUVORDERSFILENAME);
#endif
    if (DISPLAYSCREEN) printf ("%s\n", buffer);
    system (buffer);

#ifdef os9
    sprintf (buffer, "cp %s %s.backup\n", AUVORDERSFILENAME, AUVORDERSFILENAME);
#else
    sprintf (buffer, "copy %s %s.backup\n", AUVORDERSFILENAME, AUVORDERSFILENAME);
#endif
    if (DISPLAYSCREEN) printf ("%s\n", buffer);
    system (buffer);
}

    if (NOSCRIPT) /* copy default orders file for noscript mode */
    {
#ifdef os9
        sprintf (buffer, "rm %s.backup\n", AUVORDERSFILENAME);
        if (DISPLAYSCREEN) printf ("%s\n", buffer);
        system (buffer);
        sprintf (buffer, "cp %s %s.backup\n", AUVORDERSFILENAME, AUVORDERSFILENAME);
        if (DISPLAYSCREEN) printf ("%s\n", buffer);
        system (buffer);
        sprintf (buffer, "cp %s.noscript %s\n", AUVORDERSFILENAME, AUVORDERSFILENAME);

```

```

        if (DISPLAYSCREEN) printf ("%s\n", buffer);
        system (buffer);
        sprintf (buffer, "chmod +w %s\n", AUVORDERSFILENAME);
        if (DISPLAYSCREEN) printf ("%s\n", buffer);
        system (buffer);
        sprintf (buffer, "cp %s %s\n", TELEMETRYFILENAME, AUVDATAFILENAME);
        if (DISPLAYSCREEN) printf ("%s\n", buffer);
        system (buffer);
        sprintf (buffer, "cp %s %s\n", TELEMETRYFILENAME, AUVTEXTFILENAME);
        if (DISPLAYSCREEN) printf ("%s\n", buffer);
        system (buffer);
    #endif
}

/* - - - - - e-mail - - - - - */

if (replication_count <= 2) /* only send e-mail once */
{
    #ifndef os9
        sprintf (buffer, "rm %s\n", AUVEMAILFILENAME);
    #else
        sprintf (buffer, "del %s\n", AUVEMAILFILENAME);
    #endif
    if (DISPLAYSCREEN) printf ("%s\n", buffer);
    system (buffer);

    #ifndef os9
        sprintf (buffer, "cp %s %s\n", AUVINFOFILENAME, AUVEMAILFILENAME);
    #else
        sprintf (buffer, "copy %s %s\n", AUVINFOFILENAME, AUVEMAILFILENAME);
    #endif
    if (DISPLAYSCREEN) printf ("%s\n", buffer);
    system (buffer);

    #ifndef os9
        sprintf (buffer, "cat %s >> %s\n", AUVSCRIPTFILENAME, AUVEMAILFILENAME);
    #else
        sprintf (buffer, "list %s >> %s\n", AUVSCRIPTFILENAME, AUVEMAILFILENAME);
    #endif
    if (DISPLAYSCREEN) printf ("%s\n", buffer);
    system (buffer);

    if ((int) (strlen (email_address) >= 3) && (EMAIL))
    {
        sprintf (buffer, "mail %s < %s", email_address, AUVEMAILFILENAME);
    #ifndef os9
        if (DISPLAYSCREEN) printf ("%s\n", buffer);
        system (buffer);
    #else
        /* system (buffer); /* e-mail not available directly on OS-9 */
        send_buffer_to_virtual_world_socket (); /* buffer msg sent anyway */
    #endif
    }
} /* end if (replication_count <= 2) */

/* - - - - - */

/* permit changing the vehicle mission during continuous lab testing */
if (LOCATIONLAB && LOOPFOREVER)
{
    get_control_constants ();
    nextloopclock = clock () + (int) (dt * (double) CLOCKS_PER_SEC);
    record_data_on ();

    strcpy (buffer, " Load mission again");
    send_buffer_to_virtual_world_socket ();
    /* buffer containing message sent */
}

```

```

) while (LOCATIONLAB && LOOPFOREVER);/* end of lab infinite loop (if any) */
/*****
command_propellors_off (); /* all done, turn them off */

if (TRACE && DISPLAYSCREEN)
    printf ("[all done, send 'kill' message to virtual world dynamics]\n");

strcpy (buffer, "kill"); /* must start with 'shutdown' to die */
send_buffer_to_virtual_world_socket (); /*buffer containing message sent */

shutdown_virtual_world_socket (); /* close connection to virtual world */

close_device_paths ();

record_data_off ();

if (TRACE && DISPLAYSCREEN)
    printf ("[finishing main:  fflush (stdout), fflush (stderr)]\n");

fflush (stdout); /* force completion of screen write */
fflush (stderr); /* force completion of error write */

if (TRACE && DISPLAYSCREEN) printf ("[main exit:  return (0)]\n");

#ifdef os9
    if (DISPLAYSCREEN) printf ("gnuplot auv_plot_1_second.gnu\n");
    system ("gnuplot auv_plot_1_second.gnu"); /* display plotted results */
#endif

    return (0); /* main program exit */

} /* end main program block, execution is complete */

/*****
void closed_loop_control_module () /* executed each time step */
{
    double volts_per_dac = 0.046875;

    double lateralMult; /* multiple for lateral thruster voltage */
    int dt_range1, dt_range2;

    if (TRACE && DISPLAYSCREEN)
        printf ("[start closed_loop_control_module]\n");

    if ((LOCATIONLAB == FALSE) && (BENCHTEST==FALSE) && (HALTSCRIPT == FALSE))
    {
        if ((computer_voltage = read_computer_battery_voltage()) < 20.0)
        {
            HALTSCRIPT = TRUE;
            if (DISPLAYSCREEN)
                printf("Low Computer Voltage Detected:  %3.1f\n",computer_voltage);
        }
        if ((motor_voltage = read_motor_gyro_battery_voltage()) < 20.0)
        {
            HALTSCRIPT = TRUE;
            if (DISPLAYSCREEN)
                printf("Low Motor Voltage Detected:  %3.1f\n",motor_voltage);
        }
        if (leak_check ())
        {
            HALTSCRIPT = TRUE;
            if (DISPLAYSCREEN) printf("Leak Detected\n");
        }
        if (z_kal > 6.0)

```

```

    {
        HALTSCRIPT = TRUE;
        if (DISPLAYSCREEN) printf("Depth Exceeded\n");
    }
    if (DIVETRACKER && (dt_time + 30.0 <= t))
    {
        HALTSCRIPT = TRUE;
        if (DISPLAYSCREEN)
            printf("Loss of Dive Tracker for 30 Seconds\n");
    }
    if ((SONARINSTALLED) && (SONARSCANMODE == 1) &&
        (ST1000_range_kal > 0.0) && (ST1000_range_kal < 3.0))
    {
        HALTSCRIPT = TRUE;
        if (DISPLAYSCREEN)
            printf("Collision Avoidance Invoked: ST1000_range_kal = %6.3lf\n",
                ST1000_range_kal);
    }
}

if (HALTSCRIPT)
{
    execute_shutdown_script();
}

/* Read Sensors and Communicate with Virtual World *****/
if (SONARPINGED == TRUE)
{
    AUV_ST1000_range = read_sonar ();
    if ((fabs (AUV_ST1000_range - ST1000_range_kal) >= 3.0) ||
        (AUV_ST1000_range <= 0.001) ||
        (ST1000_range_kal <= 0.001))
        reset_sonar_filter = TRUE;
    kalman_sonar1000 (AUV_ST1000_range);
    SONARPINGED = FALSE;
    fprintf (st1000datafile, "%6.1lf %6.3lf %6.3lf %6.3lf %6.3lf %6.3lf\n",
        t, AUV_ST1000_range, ST1000_range_kal, AUV_ST1000_bearing,
        ST1000_range_kal * cos (radians (AUV_ST1000_bearing)),
        ST1000_range_kal * sin (radians (AUV_ST1000_bearing)));
}

control_sonar ();

speed = read_speed (); /* Added by D. Marco 1-12-96 */

rpm = (port_rpm_command + stbd_rpm_command) / 2.0;

clamp (& rpm, 700.0, -700.0, "rpm"); /* bound maximum RPM */

if (TRACE && DISPLAYSCREEN)
    printf ("[clamp (& rpm, 700.0, -700.0, \"rpm\") complete]\n");

/* Main_Motor RPM Control *****/
/* note thruster use does not preclude propeller use */
if (LOCATIONLAB) /* rpm model assumes instantaneous response */
{
    port_rpm = port_rpm_command;
    stbd_rpm = stbd_rpm_command;
}
else /* in water => propeller rpms are controlled so read actual value */
{
    port_rpm = read_port_motor_rpm ();
    stbd_rpm = read_stbd_motor_rpm ();
}

```

```

/* if using virtual world dynamics, network is source of values <<<<<<< */
    phi   = read_roll_angle    ();    /* read roll angle */
    cos_phi = cos (radians (phi));
    theta = read_pitch_angle   ();    /* read pitch angle */
    psi    = read_psi          ();    /* Read psi/heading */
    sin_psi = sin (radians (psi));
    cos_psi = cos (radians (psi));

    p      = read_roll_rate_gyro ();    /* read roll rate */
    q      = read_pitch_rate_gyro ();    /* read pitch rate */

    r      = normalize2(psi - psi_im1)/dt; /* differentiate to get r */
    psi_im1 = psi;
/*
    r      = read_yaw_rate_gyro   ();    /* Read yaw rate */

    z      = read_depth          ();    /* Read depth */
    kalman_z (z);
    if (TRACE && DISPLAYSCREEN)
        printf ("[z=%5.2f, z_kal=%5.2f]\n", z, z_kal);

    if (fabs (z_kal) < 0.0001) z_kal = 0.0;
    if (fabs (z_dot_kal) < 0.0001) z_dot_kal = 0.0;
    if (fabs (z_ddot_kal) < 0.0001) z_ddot_kal = 0.0;
    z_dot = z_dot_kal;
    w      = z_dot_kal; /* look out!! <<<< */

/* note: in laboratory using virtual world, values above are superceded */
/* estimate X and Y with Mathematical Model or Dead Reckoning */
if (LOCATIONLAB == FALSE) /* in-water, perform a valid dead reckon */
{
    XY_model_est ( (v_dls - 512) * volts_per_dac,
                   (v_drs - 512) * volts_per_dac,
                   AUV_bow_lateral, AUV_stern_lateral,
                   AUV_oceancurrent_x, AUV_oceancurrent_y, TRUE );
}
else /* virtual world providing sensor inputs */
{
    x += (speed * dt * cos_psi);
    if (fabs(x) <= 0.0001) x = 0.0; /* prevent OS-9 gasping */
    y += (speed * dt * sin_psi);
    if (fabs(y) <= 0.0001) y = 0.0; /* prevent OS-9 gasping */

    x += AUV_oceancurrent_x * dt;
    y += AUV_oceancurrent_y * dt;
}

if (TRACE && DISPLAYSCREEN)
{
    printf ("[AUV_oceancurrent_x = %3.1f,", AUV_oceancurrent_x);
    printf (" AUV_oceancurrent_y = %3.1f,", AUV_oceancurrent_y);
    printf (" AUV_oceancurrent_z = %3.1f]\n", AUV_oceancurrent_z);
}
/* Control laws **** NOTE: all k_ constants must be (+) positive **** */
#ifdef os9
/* Update Dive Tracker Ranges */
if (DIVETRACKER && (CLReaddmod(&dt_range1,&dt_range2)==NEW_DATA) &&
    (dt_range1 < 10000) && (dt_range2 < 10000) &&
    (dt_range1 > 0) && (dt_range2 > 0))
{
    divetracker_range1 = (double) dt_range1 / 12.0;
    divetracker_range2 = (double) dt_range2 / 12.0;
    dt_time = t;
}

```

```

        if ((TRACE) && (DISPLAYSCREEN))
            printf("Divetracker Ranges:  %f %f %f\n",
                t,divetracker_range1,divetracker_range2);
    }
    else
    {
        if (z_kal <= 2.0) dt_time = t;
    }
#endif

    waypoint_distance = sqrt ( (x - x_command) * (x - x_command)
                                + (y - y_command) * (y - y_command));

    /* calculate depth error OK prior to death spiral check */
    depth_error = (z_command - z_kal);

    /* constrain depth_error to +/- 15.0 feet to prevent going vertical */
    /* and enable stable pitch angle even on large depth changes */
    clamp (& depth_error, -15.0, 15.0, "depth_error"); /* feet */

    /* Zero thruster commands */
    AUV_bow_vertical = 0.0;
    AUV_stern_vertical = 0.0;
    AUV_bow_lateral = 0.0;
    AUV_stern_lateral = 0.0;
    delta_rudder = 0.0;
    delta_planes = 0.0;

    /* Recompute new thrustercommands depending on control mode *****/
    if (HOVERCONTROL) compute_hover_controls ();
    else if (TARGETCONTROL) compute_target_controls ();
    else if (WAYPOINTCONTROL) compute_waypoint_controls ();
    else if (LATERALCONTROL) compute_lateral_controls ();
    else if (ROTATECONTROL) compute_rotate_controls ();
    else if (RECOVERYCONTROL) compute_recovery_controls ();
    else if (THRUSTERCONTROL) compute_lateral_thrusters ();

    if (TRACE && DISPLAYSCREEN)
    {
        printf("Pre-sqrt thruster control calculated values:\n");
        printf("AUV_bow_vertical = %6.3f\n", AUV_bow_vertical);
        printf("AUV_stern_vertical = %6.3f\n", AUV_stern_vertical);
        printf("AUV_bow_lateral = %6.3f\n", AUV_bow_lateral);
        printf("AUV_stern_lateral = %6.3f\n", AUV_stern_lateral);
    }

    /* convert to signed sqrt to account for volts-to-thrust relationship */
    /* different multiple required between lab and auv because of polarity */
    /* discrepancy between virtual world and actual auv */
    if (LOCATIONLAB) lateralMult = 2.0;
    else lateralMult = -2.0;

    /* 2.0 * sqrt(6.0)= 4.8989 SQR 6 = 2.449 */
    AUV_bow_vertical = 4.8989 * dsign (AUV_bow_vertical )
        * sqrt (fabs (AUV_bow_vertical ));
    AUV_stern_vertical = 4.8989 * dsign (AUV_stern_vertical)
        * sqrt (fabs (AUV_stern_vertical));
    AUV_bow_lateral = lateralMult * 2.449 * dsign (AUV_bow_lateral )
        * sqrt (fabs (AUV_bow_lateral ));
    AUV_stern_lateral = lateralMult * 2.449 * dsign (AUV_stern_lateral )
        * sqrt (fabs (AUV_stern_lateral ));

    if (TRACE && DISPLAYSCREEN)
    {

```

```

        printf ("Post-sqrt thruster control calculated values:\n");
        printf ("AUV_bow_vertical   = %6.3f\n", AUV_bow_vertical);
        printf ("AUV_stern_vertical   = %6.3f\n", AUV_stern_vertical);
        printf ("AUV_bow_lateral    = %6.3f\n", AUV_bow_lateral);
        printf ("AUV_stern_lateral   = %6.3f\n", AUV_stern_lateral);
    }
    /* fins reset except in TARGETCONTROL and RECOVERYCONTROL modes */
    if ((TARGETCONTROL == FALSE) && (RECOVERYCONTROL == FALSE))
        compute_fin_controls ();

    /* constrain thruster orders +/- 24.0 volts == 3820 rpm no-load */
    /* constrain propeller orders +/- 700 rpm no-load */

    clamp (& AUV_bow_vertical,   -24.0, 24.0, "AUV_bow_vertical");
    clamp (& AUV_stern_vertical, -24.0, 24.0, "AUV_stern_vertical");
    clamp (& AUV_bow_lateral,    -24.0, 24.0, "AUV_bow_lateral");
    clamp (& AUV_stern_lateral,  -24.0, 24.0, "AUV_stern_lateral");

    clamp (& port_rpm_command, -700.0, 700.0, "port_rpm_command");
    clamp (& stbd_rpm_command, -700.0, 700.0, "stbd_rpm_command");

    /* Record Control Orders to Orders File */
    if ((NOSCRIPT == FALSE) &&
        ((HOVERCONTROL) || (TARGETCONTROL) ||
         (WAYPOINTCONTROL) || (ROTATECONTROL) ||
         (THRUSTERCONTROL)))
        fprintf (auvordersfile,
"%6.1f %6.1f %5.1f %5.1f %5.1f %6.1f %6.1f %6.1f %6.1f %5.1f %5.1f %5.1f\n",
t, psi_command, x_command, y_command, z_command,
port_rpm_command, stbd_rpm_command,
rudder_command, planes_command,
AUV_bow_vertical, AUV_stern_vertical,
AUV_bow_lateral, AUV_stern_lateral);

    /* command thruster and propellor orders */

    command_motor (AUV_bow_vertical,   BOW_VERTICAL);
    command_motor (AUV_stern_vertical, STERN_VERTICAL);
    command_motor (AUV_bow_lateral,    BOW_LATERAL);
    command_motor (AUV_stern_lateral,  STERN_LATERAL);
    command_motor (port_rpm_command,   PORT_PROP);
    command_motor (stbd_rpm_command,   STBD_PROP);

    /* Send commands to rudders and planes *****/
    command_rudder (delta_rudder);
    command_planes (delta_planes);

    /* send telemetry to tactical level and data recording files - - - - */
    record_data ();

    /* read commands from tactical level - - - - - */
    /* if (TACTICAL) read_parallel_port ();[old code] now uses socket*/

    /* update simulation clock "t" - - - - - */
    t = t + dt;

    fflush (stdout); /* force completion of screen write */
    currentloopclock = clock ();

    if (TRACE && REALTIME && DISPLAYSCREEN)
        printf ("Unused Loop Time: %5.4f\n",
            (float)(nextloopclock - currentloopclock) / (float) CLOCKS_PER_SEC);

    if ((REALTIME) &&

```



```

if (TRACE && DISPLAYSCREEN)
    printf ("\n[time_next_command = %5.1f]\n", time_next_command);

if (TRACE && DISPLAYSCREEN)
    printf ("\n[finish closed_loop_control_module ()]\n");

if (end_test)
{
    command_motor (0.0, BOW_VERTICAL);
    command_motor (0.0, STERN_VERTICAL);
    command_motor (0.0, BOW_LATERAL);
    command_motor (0.0, STERN_LATERAL);
    command_motor (0.0, PORT_PROP);
    command_motor (0.0, STBD_PROP);
}

return;

} /* end closed_loop_control_module () */

/*****
/*      Control Functions For Various Control Modes      */
*****/

void compute_hover_controls ()
{
    if (TRACE && DISPLAYSCREEN)
        printf ("[begin compute_hover_controls]\n");

    compute_vertical_thrusters ();

    /* Distant hoverpoint uses waypoint control until closer */
    if ((waypoint_distance > standoff_distance + 20.0) &&
        (detect_death_spiral () == FALSE))
    {
        if (TRACE && DISPLAYSCREEN)
            printf ("[Hoverpoint switch to WAYPOINTCONTROL]\n");

        port_rpm_command = 700;
        stbd_rpm_command = 700;
        WAYPOINTCONTROL = TRUE;
        DEADSTICKRUDDER = FALSE;
        DEADSTICKPLANES = FALSE;
        compute_waypoint_controls ();
    }

    /* close hoverpoint uses hover control */
    else
    {
        WAYPOINTCONTROL = FALSE;
        psi_command = psi_command_hover;

        /* report STABLE to tactical level once hoverpoint reached */
        if ((HOVERCONTROL) && (REPORTSTABLE) &&
            ((waypoint_distance < standoff_distance) &&
             (fabs (depth_error) < standoff_distance) &&
             (fabs (normalize2 (psi - psi_command)) < 2.5 /* degrees */)))
        {
            if ((TACTICAL) && (GPSFIXINPROGRESS == FALSE))
            {
                REPORTSTABLE = FALSE;
                strcpy (buffer, "STABLE HOVER");
                if (DISPLAYSCREEN) printf ("\n[%s]\n", buffer);
                send_buffer_to_tactical_socket (); /* message */
            }
        }
    }
}

```

```

    }

    /* Compute Control Settings */
    waypoint_angle = normalize (degrees (atan2z
                                     (y_command - y, x_command - x)));
    track_angle = normalize (waypoint_angle - psi);
    along_track_distance = cos (radians (track_angle)) *
        waypoint_distance;
    cross_track_distance = -sin (radians (track_angle)) *
        waypoint_distance;

    port_rpm_command = k_propeller_hover * along_track_distance
        - k_propeller_current * AUV_oceancurrent_x
        * cos_psi
        - k_propeller_current * AUV_oceancurrent_y
        * sin_psi
        - k_surge_hover * u;

    stbd_rpm_command = port_rpm_command;

    if (TRACE && DISPLAYSCREEN)
    {
        printf ("\nHOVERCONTROL:\n");
        printf ("psi_command = %5.1f, ", psi_command);
        printf ("x = %5.1f, y = %5.1f\n", x, y);
        printf ("waypoint_distance = %5.1f, track_angle = %5.1f\n",
            waypoint_distance, track_angle);
        printf ("along_track_distance = %5.1f, ", along_track_distance);
        printf ("cross_track_distance = %5.1f\n", cross_track_distance);
        printf ("port_rpm & stbd_rpm = %5.1f\n", port_rpm);
    }

    AUV_bow_lateral = - ( - k_thruster_psi * normalize2 (psi-psi_command)
        - k_thruster_r * r)
        + k_thruster_hover * cross_track_distance
        - k_thruster_current * AUV_oceancurrent_x
        * sin_psi
        + k_thruster_current * AUV_oceancurrent_y
        * cos_psi
        + k_sway_hover * v;

    AUV_stern_lateral = ( - k_thruster_psi * normalize2 (psi-psi_command)
        - k_thruster_r * r)
        + k_thruster_hover * cross_track_distance
        - k_thruster_current * AUV_oceancurrent_x
        * sin_psi
        + k_thruster_current * AUV_oceancurrent_y
        * cos_psi
        + k_sway_hover * v;
}

/* Extend time till next command if not at hover point yet */
if ((HOVERCONTROL) && (GPSFIXINPROGRESS == FALSE) &&
    ((waypoint_distance > standoff_distance) ||
     (fabs (depth_error) > standoff_distance) ||
     (fabs (normalize2 (psi - psi_command)) > 10.0 /* degrees */ )))
    /* cylinder test */
{
    /* still not at the hoverpoint */

```

```

        if (TRACE && DISPLAYSCREEN) printf("[HOVERCONTROL cylinder test]");
        /* continue until hoverpt reached without further script orders */
        time_next_command = t + 2.0 * dt;
    }

    if (TRACE && DISPLAYSCREEN)
        printf ("[end compute_hover_controls]\n");

    return; /* end compute_hover_controls () */
}
/*****

void compute_recovery_controls ()
{
    /* int TRACE = TRUE; */

    static double last_range_from_left = 0.0;

    double range_from_left,
           range_from_right,
           side_range_error,
           side_range_rate;

    if (TRACE && DISPLAYSCREEN)
        printf ("[Begin compute_recovery_controls]\n");

    range_from_recovery_pt -= u * dt;

    /* report STABLE to tactical level once hoverpoint reached */
    if ((REPORTSTABLE) && (range_from_recovery_pt <= 0.0))
    {
        if (TACTICAL)
        {
            REPORTSTABLE = FALSE;
            strcpy (buffer, "STABLE RECOVERY");
            if (DISPLAYSCREEN) printf ("\n[%s]\n", buffer);
            send_buffer_to_tactical_socket (); /* message */
        }
    }

    /* if both sonars are not in place, hover in place */
    if ((normalize (AUV_ST1000_bearing) > 286.0) ||
        (normalize (AUV_ST1000_bearing) < 285.0) ||
        (normalize (AUV_ST725_bearing) > 75.0) ||
        (normalize (AUV_ST725_bearing) < 74.0))
    {
        if (TRACE && DISPLAYSCREEN)
        {
            printf ("[Using hover control until sonar in place]\n");
            printf ("[AUV_ST1000_bearing %7.3lf]\n", AUV_ST1000_bearing);
            printf ("[AUV_ST725_bearing %7.3lf]\n", AUV_ST725_bearing);
        }
        compute_hover_controls ();
    }
    else
    {
        kalman_sonar725 (AUV_ST725_range);
        range_from_left = fabs (sin (radians (AUV_ST1000_bearing))
                                * ST1000_range_kal;
        range_from_right = fabs (sin (radians (AUV_ST725_bearing))
                                * ST725_range_kal;
        side_range_error = (range_from_left - range_from_right) / 2.0;

        if (NEWRECOVERYCOMMAND)
        {
            side_range_rate = 0.0;
            NEWRECOVERYCOMMAND = FALSE;
        }
    }
}

```

```

else
{
    side_range_rate = (range_from_left - last_range_from_left) / dt;
}
last_range_from_left = range_from_left;

compute_vertical_thrusters ();

/* Compute required propeller power */
port_rpm_command = k_propeller_hover * range_from_recovery_pt
                  - k_propeller_current * AUV_oceancurrent_x
                  - k_propeller_current * cos_psi
                  - k_propeller_current * AUV_oceancurrent_y
                  - k_surge_hover * u;

stbd_rpm_command = port_rpm_command;

/* Compute lateral thruster power */
AUV_bow_lateral = - ( - k_thruster_psi * normalize2 (psi-psi_command)
                    - k_thruster_r * r)
                  + k_thruster_hover * side_range_error
                  - k_thruster_current * AUV_oceancurrent_x
                  + k_thruster_current * sin_psi
                  + k_thruster_current * AUV_oceancurrent_y
                  + k_thruster_current * cos_psi
                  + k_sway_hover * side_range_rate;

AUV_stern_lateral = ( - k_thruster_psi * normalize2 (psi-psi_command)
                    - k_thruster_r * r)
                   + k_thruster_hover * side_range_error
                   - k_thruster_current * AUV_oceancurrent_x
                   + k_thruster_current * sin_psi
                   + k_thruster_current * AUV_oceancurrent_y
                   + k_thruster_current * cos_psi
                   + k_sway_hover * side_range_rate;
}

/* Extend time till next command if not at hover point yet */
if (range_from_recovery_pt > 0.0)
{
    /* still not at the recovery point */
    if (TRACE && DISPLAYSCREEN) printf("[RECOVERY cylinder test]\n");
    /* continue until recoverpt reached without further script orders */
    time_next_command = t + 2.0 * dt;
}

if (TRACE && DISPLAYSCREEN)
{
    printf ("[COMPUTED RECOVERY PARAMETERS AND CONTROLS]\n");
    printf ("[Range From Final Pt %7.3lf]\n",range_from_recovery_pt);
    printf ("[Range From Right %7.3lf]\n",range_from_right);
    printf ("[Range From LEFT %7.3lf]\n",range_from_left);
    printf ("[Range Error %7.3lf]\n",side_range_error);
    printf ("[Side Range Rate %7.3lf]\n",side_range_rate);
    printf ("[Stbd Propeller %7.3lf]\n",stbd_rpm_command);
    printf ("[Port Propeller %7.3lf]\n",port_rpm_command);
    printf ("[AUV Bow Lateral %7.3lf]\n",AUV_bow_lateral);
    printf ("[AUV Stern Lateral %7.3lf]\n",AUV_stern_lateral);
    printf ("[End compute_recovery_controls]\n");
}

```

```

    }
    return; /* end compute_recovery_controls () */
}

/*****
void compute_target_controls ()
{
    static int waiting_for_tgt_update = FALSE;

    static double cos_tgt_brg      = 0.0,
                  sin_tgt_brg      = 0.0,
                  cos_tgt_brg_cmd  = 0.0,
                  sin_tgt_brg_cmd  = 0.0,
                  commanded_psi_tgt = 0.0;

    double distance_term_factor = 1.0;

    if (TRACE && DISPLAYSCREEN)
        printf ("[Begin compute_target_controls]\n");

    if ((new_target_update) || (NEWTARGETSTATION))
    {
        cos_tgt_brg      = cos (radians (target_bearing)),
        sin_tgt_brg      = sin (radians (target_bearing)),
        cos_tgt_brg_cmd  = cos (radians (normalize
                                     (180.0 + target_bearing_command))),
        sin_tgt_brg_cmd  = sin (radians (normalize
                                     (180.0 + target_bearing_command)));

        /* Compute World Space Location of Station Point */
        x_command =      x
                      + cos_tgt_brg * target_range
                      + cos_tgt_brg_cmd * target_range_command;
        y_command =      y
                      + sin_tgt_brg * target_range
                      + sin_tgt_brg_cmd * target_range_command;

        if (NEWTARGETSTATION)
            commanded_psi_tgt = psi_command;

        NEWTARGETSTATION      = FALSE;
        if (new_target_update) waiting_for_tgt_update = FALSE;
        new_target_update      = FALSE;
    }

    if (waiting_for_tgt_update == FALSE)
    {
        if (TARGETPOINTING)
            psi_command_tgt = degrees (atan2z ((target_y - y), (target_x - x)));
        else psi_command_tgt = commanded_psi_tgt;
    }

    /* if it has been a while since the last target update, */
    /* wait here for next one */
    if ((waiting_for_tgt_update == FALSE) &&
        (((TARGETEDGETRACK) && (time_last_target_update + 5.0 <= t)) ||
         ((TARGETEDGETRACK == FALSE) && (time_last_target_update + 7.5 <= t))))
    {
        if (TRUE && DISPLAYSCREEN)
        {
            printf("Hovering Until New Target Update\n");
            printf("X: %5.1lf    Y: %5.1lf    Psi: %5.1lf\n",
                  x_command, y_command, psi_command_hover);
        }
        x_command      = x;
        y_command      = y;
        psi_command_hover = psi;
    }
}

```

```

        waiting_for_tgt_update = TRUE;
    }

    /* Re-calculate waypoint distance and depth error */
    /* to account for possibly moving target */
    waypoint_distance = sqrt ( (x - x_command) * (x - x_command)
                               + (y - y_command) * (y - y_command));

    /* If waiting for target update, use hovercontrol to hold posit */
    if (waiting_for_tgt_update)
    {
        compute_hover_controls ();
        if (REPORTSTABLE) time_next_command = t + 2.0 * dt;
        return;
    }

    waypoint_angle      = normalize (degrees (atan2z
                                              (y_command - y, x_command - x)));
    track_angle          = normalize (waypoint_angle - psi);
    along_track_distance = cos (radians (track_angle))
                           * waypoint_distance;
    cross_track_distance = - sin (radians (track_angle))
                           * waypoint_distance;

    port_rpm_command = k_propeller_hover * along_track_distance
                      - k_propeller_current * AUV_oceancurrent_x
                      * cos_psi
                      - k_propeller_current * AUV_oceancurrent_y
                      * sin_psi
                      - k_surge_hover / 2.5 * u; /* 3.0 better? */

    stbd_rpm_command = port_rpm_command;

    if (TRACE && DISPLAYSCREEN)
    {
        printf ("\nTARGETCONTROL:\n");
        printf ("Station Point: %5.1f Degrees at %5.1f Feet\n",
                target_bearing_command, target_range_command);
        printf ("psi_command = %5.1f,\n", psi_command_tgt);
        printf ("Current Point: %5.1f Degrees at %5.1f Feet\n",
                target_bearing, target_range);
        printf ("Computed Station Point: %5.1f %5.1f %5.1f\n",
                x_command, y_command, z_command);
        printf ("waypoint_distance = %5.1f, track_angle = %5.1f\n",
                waypoint_distance, track_angle);
        printf ("along_track_distance = %5.1f, ", along_track_distance);
        printf ("cross_track_distance = %5.1f\n", cross_track_distance);
        printf ("port_rpm & stbd_rpm = %5.1f\n", port_rpm);
    }

    AUV_bow_lateral = - ( - k_thruster_psi / 3.0 *
                          normalize2 (psi - psi_command_tgt)
                          - k_thruster_r * r)
                      + k_thruster_hover / 1.5
                      * cross_track_distance
                      + k_sway_hover / 2.0 * v /* 3.0 better? */
                      - k_thruster_current * AUV_oceancurrent_x
                      * sin_psi
                      + k_thruster_current * AUV_oceancurrent_y
                      * cos_psi;

    AUV_stern_lateral = ( - k_thruster_psi / 3.0 *
                        normalize2 (psi - psi_command_tgt)

```

```

        - k_thruster_r * r)
    + k_thruster_hover / 1.5
      * cross_track_distance
    + k_sway_hover / 2.0 * v /* 3.0 better? */
    - k_thruster_current * AUV_oceancurrent_x
      * sin_psi
    + k_thruster_current * AUV_oceancurrent_y
      * cos_psi;

depth_error = (z_command - z_kal);

/* constrain depth_error to +-15.0 feet to prevent going vertical */
/* and enable stable pitch angle even on large depth changes */
clamp (& depth_error, -15.0, 15.0, "depth_error"); /* feet */

compute_vertical_thrusters ();

/* If we are not at the station point yet, continue */
if ((REPORTSTABLE) &&
    ((time_last_target_update + 1.0 < t) ||
     (waypoint_distance > 0.5) ||
     (fabs (normalize2 (psi - psi_command_tgt)) > 5.0)))
{
    time_next_command = t + 2.0 * dt;
}
else
{
    if ((TACTICAL) && (REPORTSTABLE))
    {
        strcpy (buffer, "STABLE TARGET STATION");
        if (DISPLAYSCREEN) printf ("\n[%s]\n", buffer);
        send_buffer_to_tactical_socket ();
    }
    REPORTSTABLE = FALSE;
}

if (TRACE && DISPLAYSCREEN)
    printf ("[End compute_target_controls]\n");

return; /* end compute_target_controls () */
}
/*****/

void compute_waypoint_controls ()
{
    if (TRACE && DISPLAYSCREEN)
        printf ("[begin compute_waypoint_controls]\n");

    if ((port_rpm_command < 200.0) || (stbd_rpm_command < 200))
    {
        if (TRACE && DISPLAYSCREEN)
            printf ("[WAYPOINTCONTROL rpm too low, reset to 400.0]\n");
        port_rpm_command = 400.0; /* boost it higher, 200-400 are OK */
        stbd_rpm_command = 400.0;
    }

    waypoint_angle = atan2z (y_command - y + AUV_oceancurrent_y * dt,
                             x_command - x + AUV_oceancurrent_x * dt);
    waypoint_angle = normalize (degrees (waypoint_angle));
    psi_command = waypoint_angle;

    if (THRUSTERCONTROL)
        compute_lateral_thrusters ();

    /* If the auv is closer to the waypoint than this value, there */

```

```

/* is a danger that it could enter a death spiral */
death_spiral_radius = fabs (sin (radians (normalize2
                                (waypoint_angle - psi))))
                        * (rpm / 700.0) * 15.0;

if (TRACE && DISPLAYSCREEN)
{
    printf ("death_spiral_radius = %5.1f, ", death_spiral_radius);
    printf ("WAYPOINTCONTROL psi_command = %5.1f, ", psi_command);
    printf ("x = %5.1f, y = %5.1f\n", x, y);
}

/* Waypoint not reached, continue */
if ((FOLLOWWAYPOINTMODE) && (HOVERCONTROL == FALSE) &&
    (detect_death_spiral (FALSE) == FALSE) && /* check it */
    (((waypoint_distance > standoff_distance) &&
      (waypoint_distance > death_spiral_radius)) ||
     (fabs (depth_error) > standoff_distance)))
{
    if (TRACE && DISPLAYSCREEN)
        printf ("\n[FOLLOWWAYPOINTMODE cylinder test]");
    /* continue until WAYPOINT reached without further script orders */
    time_next_command = t + 2.0 * dt;
}

/* Waypoint Reached */
else if ((fabs (depth_error) <= standoff_distance) &&
         ((waypoint_distance <= standoff_distance) ||
          (waypoint_distance <= death_spiral_radius)) ||
         (detect_death_spiral (FALSE))) /* check it */
{
    WAYPOINTCONTROL = FALSE;
    FOLLOWWAYPOINTMODE = FALSE;

    if (TRACE && DISPLAYSCREEN)
        printf ("\n[FOLLOWWAYPOINTMODE success, WAYPOINT reached]");

    if (HOVERCONTROL == FALSE) DEATH_SPIRAL_RESET = TRUE;

    /* report STABLE to tactical level once waypoint received */
    if ((TACTICAL) && (REPORTSTABLE) && (HOVERCONTROL == FALSE)
        && (GPSFIXINPROGRESS == FALSE))
    {
        REPORTSTABLE = FALSE;
        strcpy (buffer, "STABLE WAYPOINT");
        if (DISPLAYSCREEN) printf ("\n[%s]\n", buffer);
        send_buffer_to_tactical_socket (); /* message */
    }
}

if (TRACE && DISPLAYSCREEN)
    printf ("[end compute_waypoint_controls]\n");

return; /* end compute_waypoint_controls () */
}
/*****

void compute_lateral_controls ()
{
    if (TRACE && DISPLAYSCREEN)
        printf ("[begin compute_lateral_controls]\n");

    compute_vertical_thrusters ();

    AUV_bow_lateral = - k_thruster_lateral * lateral_command;
    AUV_stern_lateral = AUV_bow_lateral;
}
*****/

```



```

    psi_command = psi;

    if (TRACE && DISPLAYSCREEN)
        printf ("[end compute_lateral_controls]\n");

    return;
}
/*****

void compute_rotate_controls ()
{
    if (TRACE && DISPLAYSCREEN)
        printf ("[begin compute_rotate_controls]\n");

    compute_vertical_thrusters ();

    if (TRACE && DISPLAYSCREEN)
        printf ("(ROTATECONTROL == TRUE)\n");

    AUV_stern_lateral = k_thruster_rotate * rotate_command;
    AUV_bow_lateral   = -AUV_stern_lateral; /* negative */

    if (TRACE && DISPLAYSCREEN)
        printf ("[end compute_rotate_controls]\n");
}
/*****

void compute_lateral_thrusters ()
{
    if (TRACE && DISPLAYSCREEN)
        printf ("[begin compute_lateral_thrusters]\n");

    compute_vertical_thrusters ();

    AUV_stern_lateral = - k_thruster_psi * normalize2 (psi - psi_command)
                      - k_thruster_r * r;
    AUV_bow_lateral   = - AUV_stern_lateral; /* negative */

    if (TRACE && DISPLAYSCREEN)
        printf ("[end compute_lateral_thrusters]\n");

    return;
}
/*****

void compute_vertical_thrusters ()
{
    static double depth_error_integral = 0.0;

    if (TRACE && DISPLAYSCREEN)
        printf ("[begin compute_vertical_thrusters]\n");

    if ((INTEGRALDEPTHCONTROL == FALSE) && (t >= time_int_control_on))
        INTEGRALDEPTHCONTROL = TRUE;

    /* Only use error in INTEGRAL CONTROL only, clamp to avoid saturation */
    depth_error_integral = ( depth_error_integral
                          + (z_kal - z_command) * TIMESTEP)
                          * INTEGRALDEPTHCONTROL;
    clamp (&depth_error_integral, -2.0, 2.0, "depth_error_integral");

    if (TRACE && DISPLAYSCREEN)
    {
        printf ("[z_kal = %5.3lf]\n", z_kal);
        printf ("[z_dot_kal = %5.3lf]\n", z_dot_kal);
        printf ("[z_command = %5.3lf]\n", z_command);
        printf ("[depth error = %5.3lf]\n", z_kal - z_command);
        printf ("[Integral Depth Error = %5.3lf]\n", depth_error_integral);
    }
}
/*****/

```

```

    }

    AUV_bow_vertical    = - k_thruster_z * (z_kal - z_command)
                        - k_thruster_w * z_dot_kal
                        - 5.0 * depth_error_integral;

    AUV_stern_vertical  = AUV_bow_vertical;

    if (TRACE && DISPLAYSCREEN)
        printf ("[Pre Pitch Thruster Values:  %5.3lf]\n",AUV_bow_vertical);

    /* include pitch control when hovering or tracking a target */
/* requires reverification in water, apparent sign error problem...
    if ((HOVERCONTROL) || (TARGETCONTROL) || (RECOVERYCONTROL))
    {
        AUV_bow_vertical += - k_thruster_theta * (theta - theta_command)
                           - k_thruster_theta * q * (2.0);
        AUV_stern_vertical += k_thruster_theta * (theta - theta_command)
                             + k_thruster_theta * q * (2.0);
    }
*/
    return;
}
/*****
void compute_fin_controls ()
{
    if (TRACE && DISPLAYSCREEN)
        printf ("[begin compute_fin_controls]\n");

    /* Report Stable Course to Tactical Level if Required */
    if ((REPORTSTABLE) &&
        (fabs (normalize2 (psi - psi_command)) < 2.5 /* degrees */ ))
    {
        if ((TACTICAL) && (HOVERCONTROL == FALSE) &&
            (WAYPOINTCONTROL == FALSE))
        {
            REPORTSTABLE = FALSE;
            strcpy (buffer, "STABLE COURSE");
            if (DISPLAYSCREEN) printf ("\n[%s]\n", buffer);
            send_buffer_to_tactical_socket (); /* message */
        }
    }

    /* Simplified PD rudders/planes control rules: - - - - - */
    /* calculate rudders - - - - - */

    delta_rudder = k_psi * normalize2 (psi - psi_command)
                  + (k_r * r) + (k_v * v);

    /* tanh not provided under OS-9 C, added at end of this program */
    /* tanh was change to dtanh to conserve time in approximation */
    if (SLIDINGMODECOURSE)
    {
        sigma = k_sigma_r * r + k_sigma_psi * normalize2 (psi - psi_command);
        delta_rudder = (3.1403 * r) + 81.9712 * eta_steering * dtanh (sigma);
    }

    /* reduce ordered rudder if excessive roll occurs, may work for many UUVs*/
    delta_rudder = delta_rudder * cos_phi * cos_phi;

    if (TRUE && DISPLAYSCREEN && (cos_phi * cos_phi < 0.98))
    {

```

```

        printf ("\nrudder/planes reduction factor due to roll phi = %6.3f\n",
                cos_phi * cos_phi);
    }

/* calculate planes - - - - - */
delta_planes = (k_z * depth_error)
                + (k_theta * theta) + (k_q * q) - (k_w * z_dot_kal);

if (TRACE) printf ("delta_planes=%5.1lf\n",delta_planes);
if (TRACE) printf ("depth_error =%5.1lf, product=%5.1lf\n",
                    depth_error, k_z * depth_error);
if (TRACE) printf ("theta =%5.1lf, product=%5.1lf\n",
                    theta, k_theta * theta);
if (TRACE) printf ("q =%5.1lf, product=%5.1lf\n",
                    q, k_q * q);
if (TRACE) printf ("z_command =%5.1lf, z_kal =%5.1lf\n",
                    z_command, z_kal);
if (TRACE) printf ("z_dot_kal =%5.1lf, product=%5.1lf\n",
                    z_dot_kal, -k_w * z_dot_kal);

/* temporary fix to incorrect delta_planes polarity in boat */
if (LOCATIONLAB == FALSE)
{
    if (TRACE && DISPLAYSCREEN)
        printf ("[reversing polarity delta_planes, delta_rudder]");
    delta_planes = - delta_planes;
    delta_rudder = - delta_rudder;
}

/* reduce ordered planes if excessive roll occurs, may work for many UUVs*/
delta_planes = delta_planes * cos_phi * cos_phi;

/* Dead stick means no open loop control of rudders/planes - - - - - */
if (DEADSTICKRUDDER)
{
    delta_rudder = rudder_command;
}
if (DEADSTICKPLANES)
{
    delta_planes = planes_command;
}

/* constrain planes & rudder orders +/- 22.5 degrees - do not normalize! */
clamp (& delta_rudder, -22.5, 22.5, "delta_rudder"); /* degrees */

if (fabs (rpm) < 400.0)
    clamp (& delta_planes, -22.5, 22.5, "delta_planes"); /* low speed */
else if (fabs (rpm) < 500.0)
    clamp (& delta_planes, -15.0, 15.0, "delta_planes"); /*medium speed*/
else if (fabs (rpm) < 700.0)
    clamp (& delta_planes, -10.0, 10.0, "delta_planes"); /* high speed */
else clamp (& delta_planes, -5.0, 5.0, "delta_planes"); /*super speed */

if (TRACE && DISPLAYSCREEN)
    printf ("[end compute_fin_controls]\n");

return; /* end compute_fin_controls () */
}
/*****/

/* The following four functions were added on 12 Dec 95 */
/* They are from Dave Marco's execution code and are used */
/* for speed control of the port propellers */

```

```

int port_speed_control(n_com)
    double n_com; /* revolutions per second */
{
    double Km_ls = 0.6589,
           e_n, v_ls_spc,
           eta_ls = 10.0,
           phi_ls = 5.0;

    if(fabs(n_com) < 0.25) Int_ls = 0.0;

    e_n = n_com - read_port_motor_rpm () / 60.0;
    Int_ls = Int_ls + dtanh(e_n/phi_ls)*dt;

    v_ls_spc = (1.0/Km_ls)*(n_com + eta_ls*Int_ls);

    v_dls = (int) ((1023.0/48.0)*(v_ls_spc) + 511.5);

    if(v_dls < 0 )      v_dls = 0;
    if(v_dls > 1023 )   v_dls = 1023;
    return(v_dls);
} /* end port_speed_control () */

/*****

int stbd_speed_control (n_com)
    double n_com; /* revolutions per second */
{
    double Km_rs = 0.6156,
           e_n, v_rs_spc,
           eta_rs = 10.0,
           phi_rs = 5.0;

    if(fabs(n_com) < 0.25) Int_rs = 0.0;

    e_n = n_com - read_stbd_motor_rpm() / 60.0;
    Int_rs = Int_rs + dtanh(e_n/phi_rs)*dt;

    v_rs_spc = (1.0/Km_rs)*(n_com + eta_rs*Int_rs);

    v_drs = (int) ((1023.0/48.0)*(v_rs_spc) + 511.5);

    if(v_drs < 0 )      v_drs = 0;
    if(v_drs > 1023 )   v_drs = 1023;
    return(v_drs);
} /* end stbd_speed_control () */

/*****

/* --- VERIFIED MATCH CURRENT/OPERATIONAL MARCO AUV HARDWARE/SOFTWARE --- */

double read_depth () /* Return depth in FEET */
{
    int    val      = 0;
    double new_z     = 0.0; /* zz in dave's execf.c code */
    double z_offset = 0.0;

    if (TRACE && DISPLAYSCREEN) printf ("\n[start read_depth ()]");

    if (LOCATIONLAB && DEADRECKON)
    {
        new_z = z_command;
    }
    else if (LOCATIONLAB)
    {

```

```

        new_z = z; /* no change, use virtual world value */
    }
    else /* in-water */
    {
        /* val = adc1(DEPTH_CELL_CH); */ /* Channel 7 */

        /* 0.0728 = 0.0182*4.0 */
        /* Since A/D now has 0-1023 range instead of 0-4095 */
        /* new_z = 0.0728*( (double) (val - z_val0)) + z_offset; */

        /* adc2 card has 0 - 4095 resolution */
        val = get_adc2 (DEPTH_CELL_CH, 0);
        new_z = 0.0182*( (double) (val - z_val0)) + z_offset;

        /* Calibration for Signal Amp */
        /*new_z = 0.0034285*( (double) (z_val0 - val)) + z_offset;*/
    }

    if (TRACE && DISPLAYSCREEN)
        printf ("\n[finish read_depth (), returns %5.3f]\n", new_z);

    return (new_z + depth_cell_bias);
} /* end read_depth () */

/*****

/* --- VERIFIED MATCH CURRENT/OPERATIONAL MARCO AUV HARDWARE/SOFTWARE --- */

double read_psi ()      /* return psi in degrees */
{
    unsigned short psi_bit;
    int psi_bit_int,psi_bit_old_int,delta_psi_bit;
    double angle, tpi;
    double pi = 3.1415927;

    if (TRACE && DISPLAYSCREEN) printf ("[start read_psi ()]\n");

    if (LOCATIONLAB && DEADRECKON)
    {
        angle = psi_command;
    }
    else if (LOCATIONLAB)
    {
        psi = psi; /* no change, use virtual world value */
        angle = psi; /* set up for function return */
    }
    else /* in-water */
    {
        psi_bit = Read_PortAB(0xFFFF00700);
        psi_bit &= 0x3FFF;
        psi_bit_int = psi_bit;
        psi_bit_old_int = psi_bit_old;

        delta_psi_bit = psi_bit_int - psi_bit_old_int;
        psi_bit_old = psi_bit;

        if(abs(delta_psi_bit) > 10000)
        {
            wrap_count = wrap_count - delta_psi_bit/abs(delta_psi_bit);
        }

        angle = start_psi + degrees ((read_heading () -
            dg_offset + 2.0*pi*((double) wrap_count)));

        if(fabs(angle) < 0.0001) angle = 0.0;
        /*printf("%f %f %f %d %d\n",
            angle,read_heading (),dg_offset,wrap_count,psi_bit); */
    }
}

```

```

    }

    if (TRACE && DISPLAYSCREEN)
        printf ("[finish read_psi () returns %5.3f]\n", angle);

    return (normalize (angle));
} /* end read_psi () */

/*****
/* --- VERIFIED MATCH CURRENT/OPERATIONAL MARCO AUV HARDWARE/SOFTWARE --- */
double read_roll_rate_gyro () /* Return roll rate in DEGREES/SEC
*/
{
    int    val;
    double rate;

    if (TRACE && DISPLAYSCREEN) printf ("[start read_roll_rate_gyro ()]\n");

    if (LOCATIONLAB)
    {
        rate = p; /* no change, use virtual world value */
        if (fabs (rate) < 0.0001) rate = 0.0;
    }
    else /* in-water */
    {
        val = get_adc2(ROLL_RATE_CH,0);
        /* Next two lines from old method */
        /*val = val >> 2;*/ /* Quick fix for new res */
        /*rate = (roll_rate_0/3.2113 - .31062*val)/57.295779;*/
        rate = degrees (0.07785*(roll_rate_0 - val)/57.295779);
        if(fabs(rate) < 0.0001) rate = 0.0;
    }

    rate = normalize2 (rate);

    if (TRACE && DISPLAYSCREEN)
        printf ("[finish read_roll_rate_gyro () returns %5.3f]\n", rate);

    return (rate);
} /* end read_roll_rate_gyro () */

/*****
/* --- VERIFIED MATCH CURRENT/OPERATIONAL MARCO AUV HARDWARE/SOFTWARE --- */
double read_pitch_rate_gyro () /* Return pitch rate in DEGREES/SEC
*/
{
    int    val = 0;
    double rate;

    if (TRACE && DISPLAYSCREEN) printf ("[start read_pitch_rate_gyro ()]\n");

    if (LOCATIONLAB)
    {
        rate = q; /* no change, use virtual world value */
        if (fabs (rate) < 0.0001) rate = 0.0;
    }
    else /* in-water */
    {
        val = get_adc2(PITCH_RATE_CH,0);
        /* Next two lines from old method */
        /*val = val >> 2;*/ /* Quick fix for new res */
        /*rate = (pitch_rate_0/13.69399 - .0730001*val)/57.295779;*/
        rate = degrees (0.01825*(pitch_rate_0 - val)/57.295779);
    }
}

```

```

        if(fabs(rate) < 0.0001) rate = 0.0;
    }

    rate = normalize2 (rate);

    if (TRACE && DISPLAYSCREEN)
        printf ("[finish read_pitch_rate_gyro () returns %5.3f]\n", rate);

    return (rate);
} /* end read_pitch_rate_gyro () */

/*****
/* --- VERIFIED MATCH CURRENT/OPERATIONAL MARCO AUV HARDWARE/SOFTWARE --- */
double read_yaw_rate_gyro () /* Return yaw rate in DEGREES/SEC */
{
    int    val = 0;
    double rate;

    if (TRACE && DISPLAYSCREEN) printf ("[start read_yaw_rate_gyro ()]\n");

    if (LOCATIONLAB)
    {
        rate = r; /* no change, use virtual world value */
        if (fabs (rate) < 0.0001) rate = 0.0;
    }
    else /* in-water */
    {
        /* Below for adc1 Card */
        /*val = adc1(YAW_RATE_CH);*/ /* Channel 10 */
        /*rate = 2.78* ( (double) yaw_rate_0)/13.653216 -
                                0.0732362* ( (double) val) )/57.295779;*/

        val = get_adc2(YAW_RATE_CH,0);
        /* Next two lines from old method */
        /*val = val >> 2;*/ /* Quick fix for new res */
        /*rate = 2.78*(yaw_rate_0/13.653216 - .0732362*val)/57.295779;*/
        rate = degrees (0.0509*(yaw_rate_0 - val)/57.295779);
        if(fabs(rate) < 0.0001) rate = 0.0;
    }

    rate = normalize2 (rate);

    if (TRACE && DISPLAYSCREEN)
        printf ("[finish read_yaw_rate_gyro () returns %5.3f]\n", rate);

    return (rate);
} /* end read_yaw_rate_gyro () */

/*****
/* --- VERIFIED MATCH CURRENT/OPERATIONAL MARCO AUV HARDWARE/SOFTWARE --- */
double read_port_motor_rpm () /* Reads rpm from PORT_PROP */
{
    int    pulse;
    double local_port_rpm;

    if (TRACE && DISPLAYSCREEN) printf ("[start read_port_motor_rpm ()]\n");

    local_port_rpm = read_motor (PORT_PROP);

    if (TRACE && DISPLAYSCREEN)
        printf ("[finish read_port_motor_rpm () returns %5.3f]\n",
                local_port_rpm);
}

```

```

        return (local_port_rpm);
    } /* end read_port_motor_rpm () */

/*****
/* --- VERIFIED MATCH CURRENT/OPERATIONAL MARCO AUV HARDWARE/SOFTWARE --- */
double read_stbd_motor_rpm () /* Reads rpm from STBD_PROP */
{
    int pulse;
    double local_stbd_rpm;

    if (TRACE && DISPLAYSCREEN) printf ("[start read_stbd_motor_rpm ()]\n");

    local_stbd_rpm = read_motor (STBD_PROP);

    if (TRACE && DISPLAYSCREEN)
        printf ("[finish read_stbd_motor_rpm () returns %5.3f]\n",
            local_stbd_rpm);

    return (local_stbd_rpm);
} /* end read_stbd_motor_rpm () */

/*****
/* --- VERIFIED MATCH CURRENT/OPERATIONAL MARCO AUV HARDWARE/SOFTWARE --- */
double read_motor (motor) /* Read rpm from single propellor or thruster */
    int motor;
{
/*
    motor = 0 Left Propeller . PORT_PROP RPM
              1 Right Propeller STBD_PROP RPM
              2 Bow Vertical Thruster BOW_VERTICAL volts
              3 Bow Lateral Thruster STERN_VERTICAL volts
              4 Stern Vertical Thruster BOW_LATERAL volts
              5 Stern Lateral Thruster STERN_LATERAL volts */

    int count;
    double freq,rps;
    unsigned char lobyte,hibyte;

    if (TRACE && DISPLAYSCREEN) printf ("[start read_motor ()]\n");

    if (LOCATIONLAB == FALSE) /* in water */
    {
        switch(motor)
        {
            case PORT_PROP:
                write_tim1a(3,tim_1a_control_reg,17);/* Sel Cntr 1 HOLD Reg. Card 3 */
                lobyte = read_tim1a(3,tim_1a_data_reg);
                hibyte = read_tim1a(3,tim_1a_data_reg);
                count = (int) (256*hibyte) + (int) lobyte;
                if(v_dls < 512 ) count = -count; /* Account for Direction of Rot. */
                break;

            case STBD_PROP:
                write_tim1a(3,tim_1a_control_reg,18);/* Sel Cntr 2 HOLD Reg. Card 3 */
                lobyte = read_tim1a(3,tim_1a_data_reg);
                hibyte = read_tim1a(3,tim_1a_data_reg);
                count = (int) (256*hibyte) + (int) lobyte;
                if(v_drs < 512 ) count = -count; /* Account for Direction of Rot. */
                break;

            case BOW_VERTICAL:
                write_tim1a(2,tim_1a_control_reg,17);/* Sel Cntr 1 HOLD Reg. Card 2 */
                lobyte = read_tim1a(2,tim_1a_data_reg);

```



```

        hbyte = read_timlac1(2,tim_1a_data_reg);
        count = (int) (256*hbyte) + (int) lobyte;
        if(v_dbvt < 512 ) count = -count; /* Account for Direction of Rot. */
        break;

    case STERN_VERTICAL:
        write_timla(2,tim_1a_control_reg,18); /*Sel Cntr 2 HOLD Reg. Card 2 */
        lobyte = read_timlac1(2,tim_1a_data_reg);
        hbyte = read_timlac1(2,tim_1a_data_reg);
        count = (int) (256*hbyte) + (int) lobyte;
        if(v_dbvt < 512 ) count = -count; /* Account for Direction of Rot. */
        break;

    case BOW_LATERAL:
        write_timla(2,tim_1a_control_reg,19); /*Sel Cntr 3 HOLD Reg. Card 2 */
        lobyte = read_timlac1(2,tim_1a_data_reg);
        hbyte = read_timlac1(2,tim_1a_data_reg);
        count = (int) (256*hbyte) + (int) lobyte;
        if(v_dsvt < 512 ) count = -count; /* Account for Direction of Rot. */
        break;

    case STERN_LATERAL:
        write_timla(2,tim_1a_control_reg,20); /*Sel Cntr 4 HOLD Reg. Card 2 */
        lobyte = read_timlac1(2,tim_1a_data_reg);
        hbyte = read_timlac1(2,tim_1a_data_reg);
        count = (int) (256*hbyte) + (int) lobyte;
        if(v_dslt < 512 ) count = -count; /* Account for Direction of Rot. */
        break;

    default:
        if (DISPLAYSCREEN)
            printf ("[read_motor () error: illegal motor value (%d)]\n", motor);
        break;
}

if(count != 0)
{
    freq = (1.0/count)*4.0*pow(10.0,6.0); /*F1 (1 Mhz) The 4.0 is in there */
                                         /* as a scale factor from God      */
}
else
{
    /* Sensor Not Counting */
    freq = 0.0;
}

/* 500 Counts Per Rev */
rps = (freq/500.0);
if((fabs(rps) < 1.0) || (fabs(rps) > 1000.0)) rps = 0.0;

if (TRACE && DISPLAYSCREEN)
    printf ("[finish read_motor () returns %5.3f]\n", rps);

return (rps * 60.0); /* convert from per-seconds to per-minutes */
}
else /* LOCATIONLAB == TRUE */

    return (rpm);

} /* end read_motor () */

/*****
/* --- VERIFIED MATCH CURRENT/OPERATIONAL MARCO AUV HARDWARE/SOFTWARE --- */
double read_roll_angle () /* Return roll angle in DEGREES */
{
    int    val;

```

```

double angle;

if (TRACE && DISPLAYSCREEN) printf ("[start read_roll_angle ()]\n");

if (LOCATIONLAB)
{
    angle = phi; /* no change, use virtual world value */
    if (fabs (angle) < 0.0001) angle = 0.0;
}
else /* in-water */
{
    val = get_adc2 (ROLL_ANGLE_CH,0);
    /* Next three lines from old method */
    /*val = val >> 2;*/ /* Quick fix for new res */
    /* angle = ((516.578 - val)/5.7572)/57.295779; convert to radians */
    /*angle = (-.1737*val + .1737*roll_0)/57.295779;*/
    angle = 0.043425*(roll_0 - val)/57.295779;
    if (fabs (angle) < 0.0001) angle = 0.0;
}

angle = normalize2 (angle);

if (TRACE && DISPLAYSCREEN)
    printf ("[finish read_roll_angle () returns %5.3f]\n", angle);

return (angle);
}

/*****
/* --- VERIFIED MATCH CURRENT/OPERATIONAL MARCO AUV HARDWARE/SOFTWARE --- */
double read_pitch_angle () /* Return pitch angle in DEGREES */
{
    int val;
    double angle;

    if (TRACE && DISPLAYSCREEN) printf ("[start read_pitch_angle ()]\n");

    if (LOCATIONLAB)
    {
        angle = theta; /* no change, use virtual world value */
        if (fabs (angle) < 0.0001) angle = 0.0;
    }
    else /* in-water */
    {
        val = get_adc2 (PITCH_ANGLE_CH,0);
        /* Next three lines from old method */
        /*val = val >> 2;*/ /* Quick fix for new res */
        /* angle = ((520.153 - val)/8.340)/57.295779; convert to radians */
        /*angle = ((-.1199*val + .1199*pitch_0)/57.295779);*/
        angle = degrees (0.02997*(pitch_0 - val)/57.295779);
        if (fabs (angle) < 0.0001) angle = 0.0;
    }

    angle = normalize2 (angle);

    if (TRACE && DISPLAYSCREEN)
        printf ("[finish read_pitch_angle () returns %5.3f]\n", angle);

    return (angle);
}

/*****
/* --- VERIFIED MATCH CURRENT/OPERATIONAL MARCO AUV HARDWARE/SOFTWARE --- */
double read_heading ()

```

```

/* Return heading angle with respect to local magnetic north in radians
   from directional gyro */
{
    unsigned short dg_bit;
    double         angle;

    if (TRACE && DISPLAYSCREEN) printf ("[start read_heading ()]\n");

    if (LOCATIONLAB && (DEADRECKON == FALSE))
    {
        angle = psi;
        if (fabs (angle) < 0.0001) angle = 0.0;
    }
    else if (LOCATIONLAB && (DEADRECKON))
    {
        angle = psi_command;
        if (fabs (angle) < 0.0001) angle = 0.0;
    }
    else /* in-water */
    {
        /*dg_bit = Read_PortAB(MFI_BASE);*/
        dg_bit = Read_PortAB(0xFF00700); /* why not a #define here? <<<< */
        /*dg_bit = 10000;*/
        dg_bit &= 0x3FFF;

        angle = (3.8350e-4)*((double) dg_bit);
        /*printf("Angle = %f %d\n",angle,dg_bit);*/
        /*if(fabs(angle) < 0.001) angle = 0.0;*/
    }

    angle = normalize (angle);

    if (TRACE && DISPLAYSCREEN)
        printf ("[finish read_heading () returns %5.3f]\n", angle);

    return (angle);
}

/*****
double read_speed () /* Filter the speed signal */
{
    static int old_count1,old_count2;
    static int start = TRUE;
    int count;
    unsigned char lobyte,hibyte;
    double freq;
    double avg_speed;

    if (TRACE && DISPLAYSCREEN)
        printf ("[start read_speed (), LOCATIONLAB=%d]\n", LOCATIONLAB);

    if (LOCATIONLAB)
    {
        if (TRACE && DISPLAYSCREEN)
            printf ("[finish read_speed () returns %5.3f]\n", speed);

        return (speed); /* from virtual world-paddlewheel speed = u = surge */
    }
    else if (DEADRECKON)
    {
        if (TRACE && DISPLAYSCREEN)
            printf ("[finish read_speed () DEADRECKON returns "];
        avg_speed = (speed_per_rpm * (port_rpm + stbd_rpm) / 2.0);
        if (TRACE && DISPLAYSCREEN)
            printf ("%5.3f]\n", avg_speed);
    }
}

```

```

        return (avg_speed);
    }
    else
    {
        /* I think this is Dave's speed averaging code */
        if(start)
        {
            old_count1 = 0;
            old_count2 = 0;
            start = FALSE;
        }

        write_timla(3,tim_la_control_reg,19);
        lobyte = read_timlac1(3,tim_la_data_reg);
        hibyte = read_timlac1(3,tim_la_data_reg);
        count = (int) (256*hibyte) + (int) lobyte;

        if((old_count1 == count) &&
            (old_count2 == count))
        {
            old_count1 = old_count2;
            old_count2 = count;
            return(0.0);
        }

        old_count1 = old_count2;
        old_count2 = count;
    }

    if (TRACE && DISPLAYSCREEN)
        printf ("[finish read_speed () returns %5.3f]\n", avg_speed);

    if(count != 0)
    {
        freq = (1.0/(2.0*count))* 4.0 * 10000.0; /* F3 (10,000 Hz) */
        if (freq >= 17) freq = freq * 2.0;
        else if (freq > 15) freq = freq * (1.0 + ((freq - 15.0)/2.0));
    }
    else
    {
        /* Sensor Not Counting */
        freq = 0.0;
    }

    /*Polyfit for Calibration data in marco:/vault2/marco/AUV/turbo_probe/tp.m */
    if(freq >= 4999.0)
    {
        return(fabs(speed));
    }
    else
    {
        return(0.00000973701619*freq*freq + 0.02934498907499*freq +
0.15845400316984);
    }
}
/*****
/* --- VERIFIED MATCH CURRENT/OPERATIONAL MARCO AUV HARDWARE/SOFTWARE --- */
void zero_gyro_data ()
{
    int index, val;
    int save_trace = TRACE;    /* save current TRACE value, restore later */

    if (TRACE && DISPLAYSCREEN) printf ("[start zero_gyro_data ()]\n");

    /* Marco code has a mode variable for gyro on/off, we assume always on */

```

```

if (fabs (dg_offset) < 0.001) dg_offset = 0.0;
/*z_val0      = adc1(DEPTH_CELL_CH);
yaw_rate_0    = get_adc1(YAW_RATE_CH);*/

z_val0        = get_adc2(DEPTH_CELL_CH,0);
pitch_0       = get_adc2(PITCH_ANGLE_CH,0);
roll_0        = get_adc2(ROLL_ANGLE_CH,0);
dg_offset     = read_heading ();
roll_rate_0   = get_adc2(ROLL_RATE_CH,0);
pitch_rate_0  = get_adc2(PITCH_RATE_CH,0);
yaw_rate_0    = get_adc2(YAW_RATE_CH,0);

for (i=0;i<9;++i)
{
    /*z_val0      += adc1(DEPTH_CELL_CH);
    yaw_rate_0    += get_adc1(YAW_RATE_CH);*/

    pitch_0      += get_adc2(11,0);
    roll_0       += get_adc2(12,0);
    roll_rate_0  += get_adc2(9,0);
    pitch_rate_0 += get_adc2(8,0);
    yaw_rate_0   += get_adc2(YAW_RATE_CH,0);
    dg_offset    += read_heading();
    z_val0       += get_adc2(DEPTH_CELL_CH,0);

    tsleep(5);
}

dg_offset      = dg_offset/10.0;
z_val0         = z_val0/10;
pitch_0        = pitch_0/10;
roll_0         = roll_0/10;
roll_rate_0    = roll_rate_0/10;
pitch_rate_0   = pitch_rate_0/10;
yaw_rate_0     = yaw_rate_0/10;

/*psi_bit_old = Read_PortAB(MFI_BASE);*/
psi_bit_old = Read_PortAB(0xFFFF00700);
psi_bit_old &= 0x3FFF;

if (TRACE && DISPLAYSCREEN)
{
    printf ("roll_0      = %d\n", roll_0);
    printf ("roll_rate_0 = %d\n", roll_rate_0);
    printf ("pitch_0       = %d\n", pitch_0);
    printf ("pitch_rate_0 = %d\n", pitch_rate_0);
    printf ("yaw_rate_0    = %d\n", yaw_rate_0);
    printf ("z_val0       = %d\n", z_val0);
    printf ("dg_offset    = %f\n", dg_offset);
}

if (TRACE && DISPLAYSCREEN) printf ("[finish zero_gyro_data ()]\n");

return;

} /* end zero_gyro_data () */

/*****

/* --- VERIFIED MATCH CURRENT/OPERATIONAL MARCO AUV HARDWARE/SOFTWARE --- */

void zero_surfaces ()          /* Initialize all planes & rudders to zero */
{
    if (TRACE && DISPLAYSCREEN)
        printf ("[start initialize_dacs_zero_actuators ()]\n");

    command_rudder (0.0);

```



```

/* Initialize tim_1a cards, mode = 0 init encoders only, init = 1 encoders */
/*                                     and fins                                     */

void init_tim1a(mode)
{
    int mode;

    {
        int i,j;

        if (LOCATIONLAB)
        {
            return;
        }
    }
/*
    counter 1, Card 1 - front rudder top, rear rudder bottom
    counter 2, Card 1 - front rudder bottom, rear rudder top
    counter 3, Card 1 - front plane left, rear plane right
    counter 4, Card 1 - front plane right, rear plane left
*/
    if(mode)
    {
        /* Init control surface card 1 */
        write_tim1a(1,tim_1a_control_reg,255); /* reset all board functions */
        write_tim1a(1,tim_1a_control_reg,23); /* select mastermode register */
        write_tim1a(1,tim_1a_data_reg,176); /* lobyte enables 8 bit,binary, */
                                                /* fout */
        write_tim1a(1,tim_1a_data_reg,65); /* hibernate enable fout = 1mhz etc */
        write_tim1a(1,tim_1a_control_reg,249); /* disable write prefetch */

        /* for (i=25;i<=28;i++) Use this if new chip installed */
        for (j=9;j<=12;j++) /* This is done since signal gets inverted */
            /* Counters 1-4 Only */
            {
                write_tim1a(1,tim_1a_control_reg,j); /* high output time about 8ms */
                write_tim1a(1,tim_1a_data_reg,0); /* load all hold registers */
                write_tim1a(1,tim_1a_data_reg,150); /* lobyte = 0 hibernate = 155 for */
                                                        /* 1 mhz */
            }

        for (j=1;j<=4;j++) /* Counters 1-4 Only */
        {
            write_tim1a(1,tim_1a_control_reg,j); /* program all counter mode */
                                                    /* registers see mode j */
            write_tim1a(1,tim_1a_data_reg,98); /* lobyte = reload from load */
                                                    /* or hold, count repeat */
            write_tim1a(1,tim_1a_data_reg,27); /* hibernate = nigate,count on */
                                                    /* falling edge 1mhz */
        }
    } /* End if(mode) */

    /* Init speed sensor cards 2 & 3 */

    /*
        counter 1, Card 2 - BOW VERTICAL THRUSTER SPEED
        counter 2, Card 2 - BOW LATERAL THRUSTER SPEED
        counter 3, Card 2 - STERN VERTICAL THRUSTER SPEED
        counter 4, Card 2 - STERN LATERAL THRUSTER SPEED
        counter 1, Card 3 - LEFT SCREW SPEED
        counter 2, Card 3 - RIGHT SCREW SPEED
        counter 3, Card 3 - TURBO PROBE SPEED
    */

```

```

*/
for(i=2;i<=3;++i) /* Program Master Mode Reg. for Cards 2 & 3 */
{
    write_tim1a(i,tim_1a_control_reg,0xff); /* Reset All Board Functions */
    write_tim1a(i,tim_1a_control_reg,0x17); /* Select Master Mode Reg. */
    write_tim1a(i,tim_1a_data_reg,0xb0);
    write_tim1a(i,tim_1a_data_reg,0xc1);
}

for(j=1;j<=4;++j) /* Program Counters 1-4, Card 2 */
{
    write_tim1a(2,tim_1a_control_reg,j);
    write_tim1a(2,tim_1a_data_reg,0xaa);
    write_tim1a(2,tim_1a_data_reg,203); /* Set for F1 (1Mhz) */
}

for(j=9;j<=12;++j) /* Set LOAD Reg 1-4 to Zero, Card 2 */
{
    write_tim1a(2,tim_1a_control_reg,j);
    write_tim1a(2,tim_1a_data_reg,0x00);
    write_tim1a(2,tim_1a_data_reg,0x00);
}

write_tim1a(2,tim_1a_control_reg,0x4f); /* Load Counters 1-4 Card 2 */
write_tim1a(2,tim_1a_control_reg,0x2f); /* Arm Counters 1-4 Card 2 */

write_tim1a(2,tim_1a_aux_gates_reg,0xff); /* SET AUX GATES HIGH TO WORK! */

for(j=1;j<=3;++j) /* Program Counters 1-3, Card 3 */
{
    write_tim1a(3,tim_1a_control_reg,j);
    write_tim1a(3,tim_1a_data_reg,0xaa);

    /* F1 = 203 = 0xCB = 1 Mhz
       F2 = 204 = 0xCC = 100 Khz
       F3 = 205 = 0xCD = 10 Khz
       F4 = 206 = 0xCE = 1 Kz
       F5 = 207 = 0xCF = 100 Hz
    */

    if(j==3)
    {
        /* Turbo Probe */
        write_tim1a(3,tim_1a_data_reg,205); /* Set Counter 3 for F (hz) */
    }
    else
    {
        write_tim1a(3,tim_1a_data_reg,203); /*Set Counters 1-2 for F1 (1Mhz)*/
    }
}

for(j=9;j<=11;++j) /* Set LOAD Reg 1-3 to Zero, Card 3 */
{
    write_tim1a(3,tim_1a_control_reg,j);
    write_tim1a(3,tim_1a_data_reg,0x00);
    write_tim1a(3,tim_1a_data_reg,0x00);
}

write_tim1a(3,tim_1a_control_reg,0x47); /* Load Counters 1-3 Card 3 */
write_tim1a(3,tim_1a_control_reg,0x27); /* Arm Counters 1-3 Card 3 */

write_tim1a(3,tim_1a_aux_gates_reg,0xff); /* SET AUX GATES HIGH TO WORK! */
}

void thruster_power(onoff)

```



```

/* A signal inverter has been placed between the pia card and the power
supplies for the thrusters, so in order to turn them on, bits for these
must be set low */

int onoff;
{
    if (LOCATIONLAB)
    {
        return;
    }
    switch(onoff)
    {
        case 0: /* TURN OFF */
            via0a_reg = via0a_reg | 0x3C; /* Set bits PA2-PA5 High retaining */
            /* other bits */
            via0[ORA_IRA] = via0a_reg;
            break;

        case 1:
            via0a_reg = via0a_reg & 0xC3; /* Set bits PA2-PA5 Low retaining */
            /* other bits */
            via0[ORA_IRA] = via0a_reg;
            break;
    }
}

void screw_power(onoff)
/* A signal inverter has been placed between the pia card and the power
supplies for the thrusters, so in order to turn them on, bits for these
must be set low */

int onoff;
{
    if (LOCATIONLAB)
    {
        return;
    }
    switch(onoff)
    {
        case 0: /* TURN OFF */
            via0a_reg = via0a_reg | 0x03; /* Set bits PA0-PA1 High retaining */
            /* other bits */
            via0[ORA_IRA] = via0a_reg;
            break;

        case 1:
            via0a_reg = via0a_reg & 0xFC; /* Set bits PA0-PA1 Low retaining */
            /* other bits */
            via0[ORA_IRA] = via0a_reg;
            break;
    }
}

/*****/
/*****/
/*****/

/* --- VERIFIED MATCH CURRENT/OPERATIONAL MARCO AUV HARDWARE/SOFTWARE --- */
void command_control_surface (angle, surface)
    double angle;
    int surface;
{
    /* This function sends the desired ANGLE to the specified control SURFACE
    The angle is first normalized to (-45 to 45 degrees), then correction is

```

```

        applied for the nonlinearity in the servo control module
    */
    /* Connections are:
        pin 1 = control
        pin 2 = ground
        pin 3 = 5 volts
    useful pulse widths are 600 to 2500 ms
    this program outputs positive going pulses with a 8 ms delay between pulses
    this program is set up for a 1 MHz board
    */

    int skip_pulse;
    int pulse,hipulse,lopulse,n,m,dis,larm;
    unsigned char card;

    int volt;          /* archaic */
    double a,b,c,d;    /* archaic */

    int old_pulse1 = -1; /* Init Old Pulses for cont. surface servos */
    int old_pulse2 = -1;
    int old_pulse3 = -1;
    int old_pulse4 = -1;

    if (FALSE && DISPLAYSCREEN)
        printf("[start command_control_surface ()]\n");

    if (LOCATIONLAB)
    {
        return; /* no action required in virtual world
    */
    }

    /* pulse = 39.32*angle + 5171;*/ /* angle (deg)*/
    pulse = ((int) 2252.87*angle) + 5171; /* Calib for Vehicle Servo angle
                                           (rad) */
    hipulse = pulse/256;
    lopulse = pulse - (hipulse*256);

    skip_pulse = FALSE;

    switch(surface)
    {
    case 1:
        n = 25;
        m = 233;
        dis = 0xC1;
        larm = 0x61;

        if(pulse == old_pulse1) skip_pulse = TRUE;
        old_pulse1 = pulse;
        break;

    case 2:
        n = 26;
        m = 234;
        dis = 0xC2;
        larm = 0x62;

        if(pulse == old_pulse2) skip_pulse = TRUE;
        old_pulse2 = pulse;
        break;

    case 3:
        n = 27;
        m = 235;
        dis = 0xC4;
        larm = 0x64;

        if(pulse == old_pulse3) skip_pulse = TRUE;
        old_pulse3 = pulse;

```

```

        break;

    case 4:
        n = 28;
        m = 236;
        dis = 0xC8;
        larm = 0x68;

        if(pulse == old_pulse4) skip_pulse = TRUE;
        old_pulse4 = pulse;
        break;

    default:
        if (DISPLAYSCREEN) printf("Invalid surface code\n");
        break;
}

if(!skip_pulse) /* SKIP resetting of freq out if command angle has not */
                /* changed. Otherwise servo will chatter at frequency */
                /* of control loop when command angle does not change */
{
    write_tim1a(1,tim_1a_control_reg,dis);
    write_tim1a(1,tim_1a_control_reg,n);
    write_tim1a(1,tim_1a_data_reg,lopulse);
    write_tim1a(1,tim_1a_data_reg,hipulse);
    write_tim1a(1,tim_1a_control_reg,m);
    write_tim1a(1,tim_1a_control_reg,larm);
}

if (FALSE && DISPLAYSCREEN)
    printf ("[finish command_control_surface ()]\n");

return;
} /* command_control_surface () */

/*****

/* --- VERIFIED MATCH CURRENT/OPERATIONAL MARCO AUV HARDWARE/SOFTWARE --- */

void command_rudder (angle)

/* Send angular deflection (DEGREES) to rudders.
Convention (+) angle forward rudder => auv right turn,
          (-) angle forward rudder => auv left turn */

double angle;
{
    if (TRACE && DISPLAYSCREEN) printf ("[start command_rudder ()]\n");

    /*top/bottom surfaces are slaved due to inadequate DAC card channels */
    /*positive forward rudder angle pushes bow to right => positive psi rate*/
    angle = radians (angle); /* convert degrees to radians */

    command_control_surface ( angle, BOW_RUDDER_TOP );
    /* command_control_surface ( angle, BOW_RUDDER_BOTTOM ); hardware error */
    command_control_surface (-angle, STERN_RUDDER_TOP );
    /* command_control_surface (-angle, STERN_RUDDER_BOTTOM); hardware error */

    if (TRACE && DISPLAYSCREEN) printf ("[finish command_rudder ()]\n");

    return;
} /* end command_rudder () */

*****/

```

```

/* --- VERIFIED MATCH CURRENT/OPERATIONAL MARCO AUV HARDWARE/SOFTWARE --- */
void command_planes (angle)

/* Send angular deflection (RADIANS) to bow and stern planes. Convention:
(-) bow plane angle => auv dive, (+) bow plane angle => auv rise
where auv dive = positive depth rate, auv rise = negative depth rate
*/

double angle;
{
    if (TRACE && DISPLAYSCREEN) printf ("[start command_planes ()]\n");
    /* left/right surfaces are slaved due to inadequate DAC card channels
    */
    /* positive planes angle pushes bow up, yields negative depth rate
    */

    angle = radians (angle);

    command_control_surface ( angle, BOW_PLANE_STBD );
    /* command_control_surface (-angle, BOW_PLANE_PORT );combined stern stbd */
    command_control_surface (-angle, STERN_PLANE_STBD);
    /* command_control_surface ( angle, STERN_PLANE_PORT);combined bow stbd */

    if (TRACE && DISPLAYSCREEN) printf ("[finish command_planes ()]\n");
    return;
} /* end command_planes () */

/*****

/* --- VERIFIED MATCH CURRENT/OPERATIONAL MARCO AUV HARDWARE/SOFTWARE --- */
void command_propellers_off () /* Turn off both main propellers */
{
    if (TRACE && DISPLAYSCREEN)
        printf ("[start command_propellers_off ()]\n");

    command_motor (0.0, PORT_PROP);
    command_motor (0.0, STBD_PROP);

    if (TRACE && DISPLAYSCREEN)
        printf ("[finish command_propellers_off ()]\n");

    return;
} /* end command_propellers_off () */

/*****

/* --- VERIFIED MATCH CURRENT/OPERATIONAL MARCO AUV HARDWARE/SOFTWARE --- */
void command_thrusters_off () /* Turn off both main propellers */
{
    if (TRACE && DISPLAYSCREEN)
        printf ("[start command_thrusters_off ()]\n");

    command_motor (0.0, BOW_VERTICAL);
    command_motor (0.0, STERN_VERTICAL);
    command_motor (0.0, BOW_LATERAL);
    command_motor (0.0, STERN_LATERAL);

    if (TRACE && DISPLAYSCREEN)
        printf ("[finish command_thrusters_off ()]\n");
}

```

```

        return;
    } /* end command_thrusters_off () */

    /*****
    /* --- VERIFIED MATCH CURRENT/OPERATIONAL MARCO AUV HARDWARE/SOFTWARE --- */

void command_motor (order,      motor)
    double order; int motor;
{
    /*
    motor = 0  Left  Propeller      PORT_PROP      RPM
              1  Right Propeller    STBD_PROP      RPM
              2  Bow  Vertical Thruster  BOW_VERTICAL volts
              3  Bow  Lateral Thruster  STERN_VERTICAL volts
              4  Stern Vertical Thruster BOW_LATERAL  volts
              5  Stern Lateral Thruster  STERN_LATERAL volts
    */

    /* use local variables to permit clamping without side effects */

    int  dac_value      = 0; /* range      0..1023 */
    double propellor_rpm = order; /* propellers -700.. 700 rpm */
    double thruster_volts = order; /* thrusters -24.. 24 volts */

    if (TRACE && DISPLAYSCREEN)
        printf ("[start  command_motor ()]\n");

    if ((motor == PORT_PROP) || (motor == STBD_PROP))
    {
        clamp (&propellor_rpm, -700.0, 700.0,
            "command_motor (): propellor_rpm");

        if (motor == PORT_PROP)
            dac_value = port_speed_control (propellor_rpm / 60.0);
        if (motor == STBD_PROP)
            dac_value = stbd_speed_control (propellor_rpm / 60.0);

        if (TRACE && DISPLAYSCREEN)
        {
            if (motor == PORT_PROP) printf ("[PORT ");
            else if (motor == STBD_PROP) printf ("[STBD ");
            printf ("propellor_rpm = %5.1f, dac_value = %d]\n",
                propellor_rpm, dac_value);
        }
    }
    else if ( (motor == BOW_VERTICAL) || (motor == STERN_VERTICAL)
        || (motor == BOW_LATERAL) || (motor == STERN_LATERAL) )
    {
        clamp (&thruster_volts, -24.0, 24.0,
            "command_motors (): thruster_volts");

        dac_value = (int) ((thruster_volts + 24.0) * 1023.0 / 48.0);

        if (TRACE && DISPLAYSCREEN)
            printf ("[thruster_volts = %5.1f, dac_value = %d]\n",
                thruster_volts, dac_value);
    }
    else /* erroneous motor number selected */
    {
        if (TRACE && DISPLAYSCREEN)
            printf ("[command_motor (): erroneous order/motor (%5.1f/%d)]\n",
                order, motor);

        return;
    }

    send_dac2b (dac_value, motor);

    if (TRACE && DISPLAYSCREEN)

```

```

        printf ("[finish command_motor ()]\n");
    return;
} /* end command_motor () */

/*****
/* --- VERIFIED MATCH CURRENT/OPERATIONAL MARCO AUV HARDWARE/SOFTWARE --- */
void test_alive (interval, local_start_dwell) /*no longer used by Dave? <<< */
{
    unsigned int interval;
    int local_start_dwell;

    unsigned int iinterval,jinterval;
    double test_delta;

    if (TRACE && DISPLAYSCREEN) printf ("[start test_alive ()]\n");

    local_start_dwell = local_start_dwell*100;
    interval = interval*100;
    iinterval = local_start_dwell/interval;
    jinterval = 0;
    test_delta = .4; /* Deflect 22.5 degrees */

    while(jinterval < iinterval)
    {
        command_control_surface (BOW_RUDDER_TOP, test_delta);
        tsleep(interval); /* 256ths of a second */
        test_delta = -test_delta;
        jinterval = jinterval + 1;
    }

    tsleep(200); /* 256ths of a second */

    if (TRACE && DISPLAYSCREEN) printf ("[finish test_alive ()]\n");

    return;
}

/*****
/* - NOT YET UPDATED TO CURRENT/OPERATIONAL MARCO AUV HARDWARE/SOFTWARE -- */
void get_init_avg () /* sonar ??? better name needed !!!! <<<< */
{
    int index, rng_sum;

    if (TRACE && DISPLAYSCREEN) printf ("[start get_init_avg ()]\n");

    rng_sum = 0;
    range_index = 0;

    for(index = 0; index < AVG_PTS; ++index)
    {
        via0[ORB_IRB] = (SONAR_SW1 & SONAR_SW3) | SONAR_TRIG2;
        via0[ORB_IRB] = SONAR_SW1 & SONAR_SW3;
        tsleep(5);
        range = get_adc2 (3,0);

        rng_sum += range;
        range_array[index] = range;
        ++range_index;
    }
    avg_rng = (rng_sum/AVG_PTS) * 1.0;

```

```

    if (TRACE && DISPLAYSCREEN) printf ("[finish get_init_avg ()]\n");
    return;
}

/*****
/* -- NOT YET UPDATED TO CURRENT/OPERATIONAL MARCO AUV HARDWARE/SOFTWARE -- */

void get_avg_rng ()
{
    int index, UPDATE_AVG, int_rng_sum;

    if (TRACE && DISPLAYSCREEN) printf ("[start  get_avg_rng ()]\n");

    UPDATE_AVG = 0;
    int_rng_sum = 0;

    if (((double)range > avg_rng ) ||
        (fabs((double)range - avg_rng) <= MAX_RNG_DIFF) ||
        (bad_rng >= MAX_BAD_PTS))
    {
        range_array[range_index] = range;
        ++range_index;
        UPDATE_AVG = 1;
        if(bad_rng > MAX_BAD_PTS)
        {
            ++bad_updates;
        }
        if(bad_updates >= MIN_NO_PTS)
        {
            bad_rng = 0;
        }
    }
    else
    {
        ++bad_rng;
    }

    if(UPDATE_AVG)
    {
        for(index = range_index - AVG_PTS; index <= range_index; ++index)
        {
            int_rng_sum += range_array[index];
        }

        avg_rng = int_rng_sum/AVG_PTS * 1.0;
    }
    if (TRACE && DISPLAYSCREEN) printf ("[finish get_avg_rng ()]\n");

    return;
}

/*****
/*
    Lab hardware control changes *FOLLOWING* hardware upgrade 1993:

    Telemetry to tactical level:      serial port /T1 via driver /TT
    Orders from tactical level:      parallel port /P via MFI register A
    Sonar:                            interface card device driver /T3

*/
/*****
/* --- NOT YET UPDATED TO CURRENT/OPERATIONAL MARCO AUV HARDWARE/SOFTWARE ---
*/

```

```

void open_device_paths ()
{
    if (TRACE && DISPLAYSCREEN) printf ("[start open_device_paths ()]\n");
    if (LOCATIONLAB == FALSE)
    {
        /* either /t1 serial port #1 or /tt (high baud rate driver for /
t1) */
        serialpath = open ("/t1", S_IREAD + S_IWRITE); /* get path number */
        /* /tt is device for high baud rate /t1 serial port */
        if (serialpath <= 0)
        {
            if (DISPLAYSCREEN)
            {
                printf ("open_device_paths (): unable to open serialpath /t1. ");
                printf ("Exit.\n");
            }
            exit (-1);
        }
        if (TRACE && DISPLAYSCREEN)
        printf ("[serialpath /t1 (normal baud rate) open, path number = %d]\n",
                serialpath);

        if (SONARINSTALLED)
        {
            sonarpath = open ("/t3", S_IREAD + S_IWRITE); /* get path number */
            /* /t3 is device for sonar interface card */
            if (sonarpath <= 0)
            {
                if (DISPLAYSCREEN)
                {
                    printf ("open_device_paths (): unable to open sonarpath /t3. ");
                    printf ("Exit.\n");
                }
                exit (-1);
            }
            if (TRACE && DISPLAYSCREEN)
                printf ("[sonarpath /t3 open, path number = %d]\n", sonarpath);

            tty_mode (sonarpath,1); /* initialize sonar values */
        }
        else if (TRACE && DISPLAYSCREEN)
            printf ("[sonarpath /t3 ignored, SONARINSTALLED == FALSE]\n");

        /* other paths: effectors, depth_sonar, etc. *****/
    }

    if (TRACE && DISPLAYSCREEN) printf ("[finish open_device_paths ()]\n");
    return;
}

/*****/

void tty_mode (path, mode)
    int path, mode;
{
    static struct sgbuf old,new;
    static int init = 1;
    int status;

    if (init)
    {
        init = 0;
        status = _gs_opt(path,&old);
        status = _gs_opt(path,&new);
    }
}

```



```

void read_parallel_port () /* loop and display 8 bit data from port A */
{
    static char next_char, last_char;
    static char current_command [256];
    static int index;
    unsigned char temp;

return;

    if (TRACE && DISPLAYSCREEN)
    {
        printf ("[start read_parallel_port (), ");
        if (PARALLELPORTTRACE) printf ("PARALLELPORTTRACE is ON]\n");
        else printf ("PARALLELPORTTRACE is OFF]\n");
    }
    /* see initialize_adcs () for Init_PortA & B code */

#ifdef os9
    /* Read PortA parallel port character by character for tactical orders */
    /* reference: Walt Landaker's mfi_a3.c in directory /h0/AUV and */
    /* page 3-12 of Motorola 6800 Series Manual for 6821 PIA */
    /* Programmable Interface Adapter. */

    /* Warning! You may have to reset both computers to get the parallel */
    /* port to read & write properly. Additionally, */
    /* on the 386 you can run PORTFIX to reset parallel port LPT1: */

    temp = Read_PortA ((struct MFI_PIA *) MFI_BASE); /* should clear busy! */
    index = 0;

    /* read port status (note sta not stb) */
    PortAFlag = ck_sta ((struct MFI_PIA *) MFI_BASE);

    if (PARALLELPORTTRACE && DISPLAYSCREEN)
        printf ("\n [time %5.2f read_parallel_port () resumed]", t);

    while (PortAFlag && 0x80) /* see loop break for alternate exit */
    {
        /* Note that ck_stb is used in mfi_a3 but ck_sta makes more sense */
        PortAFlag = ck_sta ((struct MFI_PIA *) MFI_BASE); /*read port status */
        last_char = next_char; /* read char and reset busy */
        next_char = Read_PortA((struct MFI_PIA *) MFI_BASE);

        if ((PortAFlag == 0x24) && (last_char == next_char)) break;

        /* if next_char changed then flag may be messed up, read anyway */
        /* check for ptr strobe */
        /* break => no character waiting */
        /* control passes outside while loop */

    else if (next_char == 13) /* CR indicates end of line */
    {
        current_command [index] = 13; /* CR /n */
        current_command [index+1] = 10; /* LF extra, not needed */
        current_command [index+2] = 0; /* end of string delimiter */
        index = 0;

        if (auvtextfile!=NULL)
        {
            fprintf (auvtextfile, "%s", current_command);
            fflush (auvtextfile); /* force completion of file write */
        }
        if (DISPLAYSCREEN)
        {
            printf ("\n\n>>> time %5.2f tactical message <<<\n", t);
            printf ("%s", current_command);
        }
    }
}

```



```

        unsigned    card;
        unsigned char    reg;
    {
        unsigned char data;

        if (TRACE && DISPLAYSCREEN) printf ("[start  read_timlac1 ()]\n");

        switch (card)
        {
            case 1:
                return(tim_lac1[reg]);
                data = tim_lac1[3];    /* Wait 2uS, one access to address (RTC) */
                break;

            case 2:
                return(tim_lac2[reg]);
                data = tim_lac2[3];    /* Wait 2uS, one access to address (RTC) */
                break;

            case 3:
                return(tim_lac3[reg]);
                data = tim_lac3[3];    /* Wait 2uS, one access to address (RTC) */
                break;

            default:
                if (DISPLAYSCREEN)
                    printf ("[read_timlac1 (): invalid card (%d)]\n", card);
                return (0);
        }

        if (TRACE && DISPLAYSCREEN)
            printf ("[finish read_timlac1 () = %d]\n", data);

        return (data); /* is this correct ?? */
    } /* end read_timlac1 () */

    /*****
    /* --- VERIFIED MATCH CURRENT/OPERATIONAL MARCO AUV HARDWARE/SOFTWARE --- */

void write_timla (card, reg, value)

    unsigned    card;
    unsigned char    reg, value;
{
    unsigned char data; /* Dummy Data for delay */

    if (LOCATIONLAB) return; /* avoid bus error writing to restricted memory */

    if (TRACE && DISPLAYSCREEN) printf ("[start  write_timla ()]\n");

    switch (card)
    {
        case 1:
            tim_lac1[reg] = value;
            data = tim_lac1[3];    /* Wait 2uS, one access address (RTC) */
            break;

        case 2:
            tim_lac2[reg] = value;
            data = tim_lac2[3];    /* Wait 2uS, one access address (RTC) */
            break;

        case 3:
            tim_lac3[reg] = value;
            data = tim_lac3[3];    /* Wait 2uS, one access address (RTC) */
    }

```

```

        break;

    default:
        if (DISPLAYSCREEN)
            printf ("[write_tim1a () error: illegal card value (%d)]\n", card);
            break;
    }
    if (TRACE && DISPLAYSCREEN) printf ("[finish write_tim1a ()]\n");

    return;
} /* end write_tim1a () */

/*****
*****
* send_dac1(s,ch) -- writes signal 's' to ada-1 dac channel 'ch'
*                  (allowable channels 0-3)
*****
*/

/* -- NOT YET UPDATED TO CURRENT/OPERATIONAL MARCO AUV HARDWARE/SOFTWARE -- */
/* no longer used... */

void send_dac1 (s,ch)
int s,ch;
{
    if (LOCATIONLAB)          /* in virtual world, do not read any slots */
    {
        return;
    }

    /*
    if (NOT_YET_REIMPLEMENTED)
    {
        ch = ch << 2;
        dac1_a[ch] = s >> 2;
        /* offset for G-96 addressing */
        /* write upper 8 bits to MSB */
        /* dac1_a[ch + DAC_LSB_OFFSET] = s << 6; /* write lower 2 bits B3,B2 */
        /* */
        return;
    } /* send_dac1 */

    *****/
    * send_dac2b (s,ch) -- writes signal 's' to dac2b dac channel 'ch'
    *                  (allowable channels 0-15)
    *****/

    /* --- VERIFIED MATCH CURRENT/OPERATIONAL MARCO AUV HARDWARE/SOFTWARE --- */

    *****/
    * dac2b(s,ch) -- writes signal 's' to dac2b dac channel 'ch'
    *                  (allowable channels 0-7)
    * s = 0    --> - 10 Vdc Output
    * s = 512  -->  0 Vdc Output
    * s = 1024 --> + 10 Vdc Output
    *
    * C
    * I
    * R          20*  *19
    * C          18*  *17
    * Channel 7 --> 16*  *15 <-- Ground
    * B Channel 6 --> 14*  *13 <-- Ground
    * R Channel 5 --> 12*  *11 <-- Ground
    * D Channel 4 --> 10*  *9  <-- Ground
    * Channel 3 -->  8*   *7  <-- Ground
    * S Channel 2 -->  6*   *5  <-- Ground

```

```

* I Channel 1 --> 4* *3 <-- Ground
* D Channel 0 --> 2* *1 <-- Ground (Bad)
* E
*
*****/

void send_dac2b (s,ch)
    int s,ch;
{
    if (LOCATIONLAB) return; /*avoid bus error writing to restricted memory */

    ch = ch << 2; /* offset for G-96 addressing */
    dac2b_a[ch] = s >> 2; /* write upper 8 bits to MSB */
    /*dac2b_a[ch + DAC_LSB_OFFSET] = s << 6;*/ /*write lower 2 bits B3,B2 */
    dac2b_a[ch + 2] = s << 6;

    return;
} /* send_dac2b */

/*****
* get_adc1(n) -- reads ada-1 adc channel 'n' (channels 0-15)
*****/

/* --- VERIFIED MATCH CURRENT/OPERATIONAL MARCO AUV HARDWARE/SOFTWARE --- */

int get_adc1 (n)
    int n;
{
    int val;

    if (LOCATIONLAB) /* in virtual world, do not read any slots */
    {
        return (0);
    }

    /*adc1_a[ADC1_CMD_REG] = n;
    while (adc1_a[ADC1_STATUS_REG] > 20);
    val = adc1_a[ADC1_MSB] << 2;
    val += adc1_a[ADC1_LSB] >> 6;*/

    adc1_a[4] = n;
    while (adc1_a[4] > 20);
    val = adc1_a[0] << 2;
    val += adc1_a[2] >> 6;

    return (val);
} /* get_adc1 */

/*****
* get_adc2(n,g); -- Reads adc-2 channel 'n' (0-15)
* with gain 'g' (0 to F => 0 - 1024)
*****/

/* --- VERIFIED MATCH CURRENT/OPERATIONAL MARCO AUV HARDWARE/SOFTWARE --- */

int get_adc2 (n,g)
    int n,g;
{
    int val;

    if (LOCATIONLAB) /* in virtual world, do not read any slots */
    {
        return (0);
    }

    /*adc2_a[ADC2_CH_GAIN] = (n << 4) | g;*/ /* set c&g, start conv */

```

```

/*while((adc2_a[ADC2_STATUS_REG] & 0x7) != 0);*/ /* wait for ready */

adc2_a[0] = (n << 4) | g; /* set c&g, start conv */
while((adc2_a[2] & 0x7) != 0); /* wait for ready */

/* This adc uses 0 - 4095 to represent full scale input, in order
to write to the dac (which uses 0 -1023 for full scale) you
must divide val by 4 or shift right by 2. Use the next line to
get full resolution.
val = adc2_a[ADC2_DATA];
The next line is used for testing purposes only

val = adc2_a[ADC2_DATA] >> 2; */

/*val = adc2_a[ADC2_DATA];*/

val = adc2_a[1];
val = val & 0x0FFF;

return (val);
} /* get_adc2 */

/*****
/*
The program code for the Multi-Function Interface originated from 'mfi.c'
Routines include Init_PortA, Init_PortB, Read_PortA, Read_PortB, Read_PortAB

Excerpt of 'mfi.c' comments follows:

Program example for the Multi-Function-Interface (MFI)
This example uses the 6821 PIA on the MFI board
General purpose functions are provided to initialize the PIA
and read/write data to the ports

MFI P2 connector definitions are provided by the GESMFI-1 data
sheet available from GESPAC, Inc.

6821 device specifics are covered in the 8-bit microprocessor
& peripheral data book from Motorola Inc.

3/1/91 J. Rawlins RealTime Software Consulting

*****/

/*****
* Init_PortA(base, dir) -- Initialize Port A of MFI
* dir: 1 = output port, 0 = input port
*****/

/* --- VERIFIED MATCH CURRENT/OPERATIONAL MARCO AUV HARDWARE/SOFTWARE --- */

void Init_PortA (base, dir)
register struct MFI_PIA *base; /* base address of MFI board on G96 bus
*/
int dir; /* direction: 1 = output port, 0 = input port */
{
register short temp;

if (LOCATIONLAB) /* in virtual world, do not read any slots */
{
return;
}
temp = (base->cra & 0x00FF); /* get current value of control A */
temp &= ~4; /* clear bit #2 so we can access ddra */
base->cra = temp;
if ( dir ) /* make port A all outputs */

```

```

        base->pra = 0x00FF;
    else
        base->pra = 0x0000; /* port A is all inputs */
    temp |= 4; /* set bit #2 to access data registers */
    base->cra = temp;
}/* Init_PortA */

/*****
 * Init_PortB(base, dir) -- Initialize Port B of MFI
 * dir: 1 = output port, 0 = input port
 *****/

/* --- VERIFIED MATCH CURRENT/OPERATIONAL MARCO AUV HARDWARE/SOFTWARE --- */

void Init_PortB (base, dir)
register struct MFI_PIA *base; /* base address of MFI board on G96 bus
*/
int dir; /* direction: 1 = output base, 0 = input base */
{
    register short temp;

    if (LOCATIONLAB) /* in virtual world, do not read any slots */
    {
        return;
    }
    temp = (base->crb & 0x00FF); /* get current value of control A */
    temp &= ~4; /* clear bit #2 so we can access ddra */
    base->crb = temp;
    if ( dir ) /* make port B all outputs */
        base->prb = 0x00FF;
    else /* port B is all inputs */
        base->prb = 0x0000;
    temp |= 4; /* set bit #2 to access data registers */
    base->crb = temp;
}/* Init_PortB */

/*****
 * Read_PortA (base) -- returns 8 bit value from port A
 *
 *****/

/* -- NOT YET UPDATED TO CURRENT/OPERATIONAL MARCO AUV HARDWARE/SOFTWARE -- */

/* not found! */

unsigned char Read_PortA (base)
register struct MFI_PIA *base; /* base address of MFI */
{
    register unsigned short temp;
    temp = base->pra; /* read data reg.should reset busy */
    return(temp & 0x00FF); /* return data to calling program */
}

/*****
 * Read_PortB (base) -- returns 8 bit value from port B
 *
 *****/

/* -- NOT YET UPDATED TO CURRENT/OPERATIONAL MARCO AUV HARDWARE/SOFTWARE -- */

/* not found! */

unsigned char Read_PortB (base)
register struct MFI_PIA *base; /* base address of MFI */
{

```



```

        register unsigned short temp;

    if (LOCATIONLAB)          /* in virtual world, do not read any slots */
    {
        return (0);
    }
    temp = base->prb;
    return(temp & 0x00FF);
}

/*****
 * Read_PortAB (base) -- return a 16 bit value from ports
 *                      A and B combined then mask off
 *                      the 15 th and 16 th bits.
 * Note: PIA PA0-PA7 is the LSB and PB0-PB7 the MSB
 *****/

/* --- VERIFIED MATCH CURRENT/OPERATIONAL MARCO AUV HARDWARE/SOFTWARE --- */

unsigned short Read_PortAB (base)
register struct MFI_PIA *base;      /* base address of MFI */
{
    register unsigned short hi,lo,temp;

    if (LOCATIONLAB)          /* in virtual world, do not read any slots */
    {
        return (0);
    }
    lo = (base->pra & 0x00FF); /* get least significant byte from A */
    hi = (base->prb & 0x00FF); /* and most significant byte from B */
    temp = ((hi << 8) + lo); /* shift hi into upper byte of word */
    return ( temp );         /* return data */

/*****
/* -- NOT YET UPDATED TO CURRENT/OPERATIONAL MARCO AUV HARDWARE/SOFTWARE -- */
/* not found! */

void set_bsyA(base) /* sets CB2 high (for busy to sending port) */
register struct MFI_PIA *base; /* base address of MFI */
{
    register short temp;

    if (LOCATIONLAB)          /* in virtual world, do not read any slots */
    {
        return;
    }
    temp = (base->cra & 0xFF); /* save cra values */
    base->cra = 0x38;          /* 8 bit 1= CR2 high */
    base->cra = temp;          /* restore cra values */

/* -- NOT YET UPDATED TO CURRENT/OPERATIONAL MARCO AUV HARDWARE/SOFTWARE -- */
/* not found! */

/* sets CB2 low (for -busy to sending port) */
void rst_bsyA(base)
register struct MFI_PIA *base; /* base address of MFI */
{
    register short temp;

    if (LOCATIONLAB)          /* in virtual world, do not read any slots */
    {
        return;

```

```

    }
    temp = (base->cra & 0xFF); /* save cra values */
    base->cra = 0x30; /* 8 bit 0= CR2 low */
    base->cra = temp; /* restore cra values */
}

/* -- NOT YET UPDATED TO CURRENT/OPERATIONAL MARCO AUV HARDWARE/SOFTWARE -- */

/* not found! */

int ck_sta (base)
register struct MFI_PIA *base; /* base address of MFI */
{
    register unsigned short temp;

    if (LOCATIONLAB) /* in virtual world, do not read any slots */
    {
        return (0);
    }
    temp = base->cra; /* save cra values */
    return (temp);
}

/*****

char send_sonar_command (command_char)
    char command_char;
{
    /* This function sends a single character and reads back a single
    character from the sonar */

    unsigned n,n_bytes;
    char reply,sonar_read[20],command[1];
    command[0] = command_char;

    if (TRACE && DISPLAYSCREEN)
        printf ("[Begin send_sonar_command ()]\n");

    n = write(sonarpath,command,1);
    tsleep(3);
    n_bytes = _gs_rdy(sonarpath);

    if(n_bytes == 1)
    {
        n = read(sonarpath,sonar_read,n_bytes);
    }
    reply = sonar_read[0];

    if (TRACE && DISPLAYSCREEN)
        printf ("[End send_sonar_command ()]\n");

    return(reply);
} /* end send_sonar_command () */

*****/

void set_step_size(step_code)
    int step_code;
{
    unsigned n,n_bytes;
    char reply,sonar_read[20],command[1];

    if (TRACE && DISPLAYSCREEN)
        printf ("[Begin set_step_size]\n");

    switch(step_code)
    {

```

```

        case 1:
            command[0] = 'H'; /* 0.9 degrees for st725 I don't know for st1000 */
            SONARHEADINGSTEP = 0.9;
            break;

        case 2:
            command[0] = '1'; /* 1.8 degrees for st1000 I think Y for st725 */
            SONARHEADINGSTEP = 1.8;
            break;

        case 4:
            command[0] = '2'; /* 3.6 degrees */
            SONARHEADINGSTEP = 3.6;
            break;
    }

    if(step_code == 0)
    {
        printf("***** Step Code Is 0 *****\n");
    }
    else
    {
        write(sonarpath,command,1);
        tsleep(5);
        n_bytes = _gs_rdy(sonarpath);

        if(n_bytes == 1)
        {
            n = read(sonarpath,sonar_read,n_bytes);
        }
    }

    if (TRACE && DISPLAYSCREEN) printf ("[End set_step_size]\n");

    return;
} /* end set_step_size () */

/*****
void initialize_sonar ()
{
    int TRACE = TRUE;

    int gain      = 25; /* value from 0 to 100 */
    int max_range = 10; /* others: 1,2,4,6,10,20,25,30,50,100 meters */
    unsigned n_bytes;
    unsigned short T1,T2,T3,Tchecksum;
    char command[1],sonar_read[20],*t,sonar_write[1],reply;
    unsigned short byte,byte1,Nsampl,Nbins,Range_Code,checksum;
    unsigned short TxPulseMSByte,TxPulseLSByte,Gecmin,Rng_unt;
    int i,j,k,n,word,TxPulse;

    int Timeout,Lokout,Eswait,Gaindt,Ecsclx,Ecsclx;
    int Maxdst,Dacscx,Dacscy;
    int EchoSounder[10];

    if (SONARINSTALLED == FALSE) return;
    if (TRACE && DISPLAYSCREEN)
    {
        printf("[Begin initialize_sonar]\n");
        printf("Sonar max_range = %d\n",max_range);
        printf("Sonar gain = %d\n",gain);
    }
}

```

```

/*
  INITIALIZATION PARAMETERS FOR PROFILING MODE (ST-1000 HEAD)

```

Range meters	TxPulse	NSAMPL	NBINS	Range Code	TIMOUT	Maxdst
1	30	1	64	00	1500	1500
2	30	1	64	01	3000	3000
4	30	1	128	02	6000	6000
6	30	1	128	03	9000	9000
10	40	1	128	04	15000	15000
20	50	3	128	05	30000	30000
30	75	6	128	06	45000	45000
50	100	12	128	07	65535	65535

```

ECPULS = 30
LOKOUT = 200
ESWAIT = 25600
GECMIN = Byte
GAINDT = 64
ECSCLX = 16383
ECSCLY = 11374
DACSCX = 256
DACSCY = 3125
Rng Unt = 1

```

above for all ranges.

```

*/
switch(max_range)
{
  case 1:
    TxPulse   = 30;
    Nsampl    = 1;
    Nbins     = 64;
    Range_Code = 0;

    Timeout   = 1500;
    Maxdst    = 1500;
    break;

  case 2:
    TxPulse   = 30;
    Nsampl    = 1;
    Nbins     = 64;
    Range_Code = 1;

    Timeout   = 3000;
    Maxdst    = 3000;
    break;

  case 4:
    TxPulse   = 30;
    Nsampl    = 1;
    Nbins     = 128;
    Range_Code = 2;

    Timeout   = 6000;
    Maxdst    = 6000;
    break;

  case 6:
    TxPulse   = 30;
    Nsampl    = 1;
    Nbins     = 128;
    Range_Code = 3;

    Timeout   = 9000;
    Maxdst    = 9000;

```

```

        break;

    case 10:
        TxPulse    = 40;
        Nsampl     = 1;
        Nbins      = 128;
        Range_Code  = 4;

        Timeout    = 15000;
        Maxdst     = 15000;
        break;

    case 20:
        TxPulse    = 50;
        Nsampl     = 3;
        Nbins      = 128;
        Range_Code  = 5;

        Timeout    = 30000;
        Maxdst     = 30000;
        break;

    case 30:
        TxPulse    = 75;
        Nsampl     = 6;
        Nbins      = 128;
        Range_Code  = 6;

        Timeout    = 45000;
        Maxdst     = 45000;
        break;

    case 50:
        TxPulse    = 100;
        Nsampl     = 12;
        Nbins      = 128;
        Range_Code  = 7;

        Timeout    = 65535;
        Maxdst     = 65535;
        break;
    default:
        break;
} /* End switch */

/* Values Common to all Ranges */
Lokout    = 200;
Eswait    = 25600;
Gecmin    = 2.55*gain;

Gaindt    = 64;
Ecsclyx   = 16383;
Ecscly    = 11374;
Dacscx    = 256;
Dacscy    = 3125;
Rng_unt    = 1;

/* Send Sonar Parameters */
command[0] = 'P';
write(sonarpath,command,1);

/* Send TxPulse Length (Word) in 1.96 usec units */
word = TxPulse & 0x00ff; /* Byte = LSByte of TxPulse */
byte = word;
TxPulseLSByte = byte;

sonar_write[0] = (char) byte;
n = write(sonarpath,sonar_write,1); /* Send LSByte First */

```

```

word = TxPulse >> 8;    /* Byte = MSByte of TxPulse */
byte = word;
TxPulseMSByte = byte;
sonar_write[0] = (char) byte;
n = write(sonarpath, sonar_write, 1);    /* Send MSByte Last */

/* Send NSAMPL (Byte) NO. A/D Samples per Bin */
byte = Nsampl;
sonar_write[0] = (char) byte;
n = write(sonarpath, sonar_write, 1);

/* Send NBINS (Byte) No. of Bins to Collect */
byte = Nbins;
sonar_write[0] = (char) byte;
n = write(sonarpath, sonar_write, 1);

/* Send Range Code 0-8 (obsolete) (Byte) */
byte = Range_Code;
sonar_write[0] = (char) byte;
n = write(sonarpath, sonar_write, 1);

/* Send DataByte checksum (Byte). Should be the lowest 8 bits of */
/* the sum of all Bytes */
word = TxPulseMSByte + TxPulseLSByte + Nsampl + Nbins + Range_Code;
word = word & 0x00ff; /* Mask MSByte to get last 8 bits for checksum; */
checksum = word;
/*printf("checksum = %d\n", checksum);*/
byte = checksum;
sonar_write[0] = (char) byte;
n = write(sonarpath, sonar_write, 1);

tsleep(5);
n_bytes = _gs_rdy(sonarpath);

/* Read Reply to Checksum */
n = read(sonarpath, sonar_read, 1);
reply = sonar_read[0];
if(reply == 'T')
{
    if (TRACE && DISPLAYSCREEN) printf("Sonar Parameter Checksum Ok\n");
}
else
{
    if (TRACE && DISPLAYSCREEN)
        printf("Sonar Parameter Checksum INCORRECT!!!\n");
}

/* Enable TVG should reply 'X' */
reply = send_sonar_command('X');
if(reply == 'X')
{
    if (TRACE && DISPLAYSCREEN) printf("Sonar TVG set\n");
}
else
{
    if (TRACE && DISPLAYSCREEN) printf("Sonar TVG not set!\n");
}

/* Set mode return Range bin Peak should reply 'K' */
reply = send_sonar_command('K');
if(reply == 'K')
{
    if (TRACE && DISPLAYSCREEN) printf("Sonar Range bin Peak mode Ok\n");
}
else
{
    if (TRACE && DISPLAYSCREEN)

```

```

        printf("Sonar Range bin Peak mode not set!\n");
    }

    /* Set Final Gain for TVG, should reply 'E' */
    reply = set_scanning_gain(83, 'E');
    if(reply == 'E')
    {
        if (TRACE && DISPLAYSCREEN) printf("Sonar Final TVG Gain set\n");
    }
    else
    {
        if (TRACE && DISPLAYSCREEN) printf("Sonar Final TVG Gain not set!\n");
    }

    EchoSounder[0] = 75;
    EchoSounder[1] = Timeout;
    EchoSounder[2] = Lokout;
    EchoSounder[3] = Eswait;
    EchoSounder[4] = Gaindt;
    EchoSounder[5] = Ecsclx;
    EchoSounder[6] = Ecscly;
    EchoSounder[7] = Maxdst;
    EchoSounder[8] = Dacscx;
    EchoSounder[9] = Dacscy;

    checksum = 0;

    /* Send Profiler Sonar Parameters */
    command[0] = 'J';
    write(sonarpath, command, 1);

    /* Send First 4 Parameters (Words) */
    for(i=0; i<4; ++i)
    {
        word = EchoSounder[i] & 0x00ff; /* Byte = LSByte of EchoSounder[i] */
        byte = word;
        checksum = checksum + byte; /* Add up the checksum */
        sonar_write[0] = (char) byte;
        n = write(sonarpath, sonar_write, 1); /* Send LSByte First */

        word = EchoSounder[i] >> 8; /* Byte = MSByte of EchoSounder[i] */
        byte = word;
        checksum = checksum + byte; /* Add up the checksum */
        sonar_write[0] = (char) byte;
        n = write(sonarpath, sonar_write, 1); /* Send MSByte Last */
    }

    /* Send Gecmin (Byte) */
    byte = Gecmin;
    checksum = checksum + byte;
    sonar_write[0] = (char) byte;
    n = write(sonarpath, sonar_write, 1);

    /* Send Last 6 Parameters (Words) */
    for(i=4; i<10; ++i)
    {
        word = EchoSounder[i] & 0x00ff; /* Byte = LSByte of EchoSounder[i] */
        byte = word;
        checksum = checksum + byte; /* Add up the checksum */
        sonar_write[0] = (char) byte;
        n = write(sonarpath, sonar_write, 1); /* Send LSByte First */

        word = EchoSounder[i] >> 8; /* Byte = MSByte of EchoSounder[i] */
        byte = word;
        checksum = checksum + byte; /* Add up the checksum */
        sonar_write[0] = (char) byte;
        n = write(sonarpath, sonar_write, 1); /* Send MSByte Last */
    }

```

```

    }

    /* Send Rng_uint (Byte) */
    byte = Rng_uint;
    checksum = checksum + byte;
    sonar_write[0] = (char) byte;
    n = write(sonarpath, sonar_write, 1);

    /* Send DataByte checksum (Byte). Should be the lowest 8 bits of */
    /* the sum of all Bytes */
    checksum = checksum & 0x00ff; /* Mask MSByte to get last 8 bits */

    if (TRACE && DISPLAYSCREEN)
        printf("Sonar Profile checksum = %d\n", checksum);
    byte = checksum;
    sonar_write[0] = (char) byte;
    n = write(sonarpath, sonar_write, 1);

    tsleep(5);
    n_bytes = _gs_rdy(sonarpath);

    /* Read Reply to Checksum */
    n = read(sonarpath, sonar_read, 1);
    reply = sonar_read[0];
    if(reply == 'T')
    {
        if (TRACE && DISPLAYSCREEN) printf("Sonar Profile Checksum Ok\n");
    }
    else
    {
        if (TRACE && DISPLAYSCREEN) printf("Sonar Profile Checksum Incorrect\n");
    }

    /* Check if head is using default settings. Reply is 'T' if yes, */
    /* 'F' if not */
    reply = send_sonar_command('D');
    if(reply == 'T')
    {
        if (TRACE && DISPLAYSCREEN)
            printf("Sonar Head is still using Default Settings!\n");
    }
    else
    {
        if (TRACE && DISPLAYSCREEN)
            printf("Sonar Head not using Default Settings\n");
    }

    if (TRACE && DISPLAYSCREEN) printf("[End initialize_sonar]\n");
    return;
}

/*****
char set_scanning_gain(gain, which_gain)

    int gain;
    char which_gain; /* which_gain = 'B' for Initial, 'E' for Final */
{
    int TRACE = TRUE;

    unsigned short byte;
    unsigned n, n_bytes;
    char reply, sonar_write[1], sonar_read[20], command[1];

    if (TRACE && DISPLAYSCREEN)
        printf (" [Begin set_scanning_gain]\n");

```



```

    /* Set Initial or Final Gain for TVG should reply */
    /* 'C' for Initial or 'E' for Final */
    command[0] = which_gain;
    write(sonarpath,command,1);
    byte = 2.55*gain;
    sonar_write[0] = (char) byte;
    n = write(sonarpath,sonar_write,1);
    tsleep(5);
    n_bytes = _gs_rdy(sonarpath);
    read(sonarpath,sonar_read,n_bytes);
    reply = sonar_read[0];

    if (TRACE && DISPLAYSCREEN)
        printf ("[End set_scanning_gain]\n");

    return(reply);
}

/*****
void center_sonar ()
{
    int TRACE = TRUE;

    int n, n_bytes, encoder_width, count;
    char encode;
    char sonar_read[200];

    if (SONARINSTALLED == FALSE) return;

    if (TRACE && DISPLAYSCREEN)
        printf ("[Begin center_sonar]\n");

    AUV_ST1000_bearing = 0.0;

    if (LOCATIONLAB == FALSE)
    {
        encoder_width = 0;
        n_bytes=_gs_rdy(sonarpath);

        /* Flush the Buffer to Ensure a Clean Start */
        if(n_bytes != -1)
        {
            if (TRACE && DISPLAYSCREEN)
                printf("n_bytes in sonar buffer = %d\n",n_bytes);
            n=read(sonarpath,sonar_read,n_bytes);
        }

        /* Clear out any junk from buffer at startup */
        while(encode != '3')
        {
            encode = send_sonar_command('V');
        }

        /* Get Outside Encoder */
        do {
            encode = send_sonar_command('+');
        } while ((encode != 'f') && (encode != 'F'));
        if (DISPLAYSCREEN) printf("[Found ST1000 Non-Encoder Space]\n");

        /* Find Encoder Edge */
        do {
            encode = send_sonar_command('-');
        } while ((encode != 't') && (encode != 'T'));
        encode = send_sonar_command('+');
        if (DISPLAYSCREEN) printf("[Found ST1000 Encoder Edge]\n");
    }
}

```

```

    /* Sweep Across Encoder To Determine Width */
    encoder_width = 0;
    do {
        encode = send_sonar_command('-');
        encoder_width++;
    } while ((encode != 'f') && (encode != 'F'));
    if (DISPLAYSCREEN)
        printf("[ST1000 Encoder Width Determined: %d]\n",encoder_width);

    /* Sweep Back Halfway Into Encoder */
    for (count = 0; count < encoder_width / 2; count++)
        encode = send_sonar_command('+');
    if (DISPLAYSCREEN) printf("[ST1000 Center Established]\n");
}

if (TRACE && DISPLAYSCREEN) printf ("[End center_sonar]\n");

return;
}
/*****

void step_sonar (direction)
    int direction;
{
    if (SONARINSTALLED == FALSE) return;

    if (TRACE && DISPLAYSCREEN)
        printf ("[Begin step_sonar]\n");

    if (direction == LEFT)
        AUV_ST1000_bearing = normalize (AUV_ST1000_bearing - SONARHEADINGSTEP);
    else if (direction == RIGHT)
        AUV_ST1000_bearing = normalize (AUV_ST1000_bearing + SONARHEADINGSTEP);

    if ((AUV_ST1000_bearing < 0.9) || (AUV_ST1000_bearing > 359.1))
        AUV_ST1000_bearing = 0.0;

    if (TRACE && DISPLAYSCREEN)
    {
        printf ("[Step Direction: %d]\n",direction);
        printf ("[New ST1000 Bearing: %6.2lf]\n",AUV_ST1000_bearing);
    }

    if (TRACE && DISPLAYSCREEN)
        printf ("[End step_sonar]\n");
}

/*****

void ping_sonar (direction)
    int direction;
{
    int i,n,n_bytes;
    char command[1],sonar_read[20];

    if (SONARINSTALLED == FALSE) return;

    if (TRACE && DISPLAYSCREEN)
        printf ("[Begin ping_sonar]\n");

    if (LOCATIONLAB == FALSE)
    {
        /* Clear Out Any Data That May be On Port Prior to Write */
        n_bytes = _gs_rdy(sonarpath);
        if(n_bytes != -1)
        {
            if (TRACE && DISPLAYSCREEN)
                printf("[Clearing %d junk bytes before write to sonar]\n",n_bytes);

```

```

        n=read(sonarpath,sonar_read,n_bytes);
    }

    if(direction == 0)
    {
        if (TRACE && DISPLAYSCREEN)
            printf("[Sonar Ping Only]\n");
        command[0] = 'Z';
        /* Global Variable that indicates how many bytes to expect */
        st1000_bytes_expected = 2;
    }
    else if( (direction == LEFT) || (direction == RIGHT) )
    {
        if (TRACE && DISPLAYSCREEN)
        {
            if (direction == LEFT)
                printf("[Ping sonar and step left]\n");
            else printf("[Ping sonar and step right]\n");
            /* ST1000 mounted upside down on bottom, left & right reversed */
            if(direction == LEFT) command[0] = ')';
            if(direction == RIGHT) command[0] = '(';
            /* Global Variable that indicates how many bytes to expect */
            st1000_bytes_expected = 3;
        }
    }
    else
    {
        printf("[Invalid direction, command ignored!]\n");
        return;
    }

    if (TRACE && DISPLAYSCREEN)
        printf("[Performing write to sonar]\n");
    write(sonarpath, command, 1);
}

if ((direction == LEFT) || (direction == RIGHT) || (direction == 0))
{
    SONARPINGED = TRUE;
    step_sonar (direction);
}

if (TRACE && DISPLAYSCREEN)
    printf ("[End ping_sonar]\n");

return;

} /* end ping_sonar () */

/*****

double read_sonar ()
{
    /*  int TRACE = TRUE; */

    int i,n,n_bytes;
    int RESET_PORT;
    char sonar_read[200],sonar_read2[1],*ttt;
    unsigned short profilebyte;
    unsigned short timeout;

    if (SONARINSTALLED == FALSE) return 0.0;

    if (TRACE && DISPLAYSCREEN)
        printf ("[Begin read_sonar]\n");

    if (LOCATIONLAB == FALSE)
    {
        ttt = (char *) & profilebyte;

```

```

n_bytes = _gs_rdy(sonarpath);

timeout = 0;
RESET_PORT = FALSE;

/* Loop Until Data is There */
while(n_bytes < st1000_bytes_expected)
{
    tsleep(2);
    n_bytes = _gs_rdy(sonarpath);
    if (TRACE && DISPLAYSCREEN)
        printf("ST1000 bytes ready = %d\n",n_bytes);
    if((timeout == 2) && (n_bytes != st1000_bytes_expected))
    {
        RESET_PORT = TRUE;
        break;
    }
    timeout++;
}

if(RESET_PORT)
{
    if (TRACE && DISPLAYSCREEN) printf("[***Sonar Port Reset***]\n");
    AUV_ST1000_range = 0.0;
    tty_mode(sonarpath,0);
    close(sonarpath);
    sonarpath = open("/t3",S_IWRITE+S_IREAD);
    tty_mode(sonarpath,1);
    n_bytes = _gs_rdy(sonarpath);
    if(n_bytes != -1 ) n=read(sonarpath,sonar_read,n_bytes);/*Read Junk */
    n_bytes = -1;
}

if(n_bytes == st1000_bytes_expected)
{
    if (TRACE && DISPLAYSCREEN)
        printf("ST1000 correct # of bytes = %d\n",n_bytes);

    n=read(sonarpath,sonar_read,1);
    ttt[1] = sonar_read[0];
    n=read(sonarpath,sonar_read,1);
    ttt[0] = sonar_read[0];
    if(st1000_bytes_expected == 3) n=read(sonarpath,sonar_read2,1);
    /*Read TtFf*/
    AUV_ST1000_range = 0.0328*profilebyte;
}

if(n_bytes > st1000_bytes_expected)
{
    /* Incorrect number of bytes in buffer, clear it out! */
    if (TRACE && DISPLAYSCREEN)
        printf("[***ST1000 Incorrect # of bytes = %d***]\n",n_bytes);

    n=read(sonarpath,sonar_read,n_bytes);
    AUV_ST1000_range = 0.0;
}

}

if (AUV_ST1000_range > 32.808) AUV_ST1000_range = 0.0;

if (TRACE && DISPLAYSCREEN)
{
    printf (" [AUV_ST1000_range = %5.3lf]\n",AUV_ST1000_range);
    printf (" [End read_sonar]\n");
}

return(AUV_ST1000_range);

```

```

)

/*****
void control_sonar ()      /* ST1000 execution level (ST725 is tactical level) */
{
    static double previous_range      = 0.0;
    static double start_bearing       = 0.0;
    static double end_bearing         = 0.0;
    static double range_accumulator   = 0.0;
    static int    valid_return_count  = 0;
    static int    no_return_count     = 0;
    static int    scan_onto_target    = FALSE;
    static int    update_target_data  = FALSE;

    double new_target_bearing,
           new_target_range,
           commanded_bearing_error,
           sonar_return_x,
           sonar_return_y;

    if (SONARINSTALLED == FALSE) return;

    if (TRACE && DISPLAYSCREEN)
        printf ("[Begin control_sonar]\n");

    /*****
    /* Sonar Scan Modes
    /*
    /* 1: normal forward scan (SCANWIDTH/2.0 degrees either side)
    /* 2: target edge scan (either left or right)
    /* 3: full target scan
    /* 4: locate target
    /* 5: manually position
    /*
    /*****
    switch (SONARSCANMODE)
    {
        case (1): /* Normal Straight Ahead Scan Mode to Detect Collisions */
            if (TRACE && DISPLAYSCREEN)
                printf ("[Sonar using normal forward scan]\n");

            /* See if it is time to reverse scan direction */
            if (((SONARSCANDIRECTION == RIGHT) || (SONARSCANDIRECTION == 0)) &&
                (normalize2 (AUV_ST1000_bearing) >= normalize2 (SCANWIDTH/2.0)))
                SONARSCANDIRECTION = LEFT;
            else if (((SONARSCANDIRECTION == LEFT) || (SONARSCANDIRECTION == 0)) &&
                (normalize2 (AUV_ST1000_bearing) <= -normalize2 (SCANWIDTH/2.0)))
                SONARSCANDIRECTION = RIGHT;
            else if (SONARSCANDIRECTION == 0)
                SONARSCANDIRECTION = RIGHT;

            ping_sonar (SONARSCANDIRECTION);

            break;

        case (2): /* Edge Tracking Mode */
            if (TRACE && DISPLAYSCREEN)
                printf ("[Sonar in edge tracking mode]\n");

            /* Decide whether or not ping was on the target or not */
            if ((ST1000_range_kal > 0.0) && /* fabs needed next line ?? */
                (fabs (ST1000_range_kal - previous_range) < 3.0))
            {
                if (((valid_return_count == 0) && (scan_onto_target)) ||
                    (scan_onto_target == FALSE))

```

```

        {
            start_bearing    = normalize (psi + AUV_ST1000_bearing);
        }
        previous_range      = ST1000_range_kal;
        no_return_count     = 0;
        range_accumulator   += ST1000_range_kal;
        valid_return_count++;
    }
    else
    {
        no_return_count++;
    }

    /* Check to see if it is time to reverse the scan and compute numbers */
    if (scan_onto_target)
    {
        if ((valid_return_count > 2) ||
            ((valid_return_count > 0) && (no_return_count > 2)))
        {
            update_target_data = TRUE;
            scan_onto_target   = FALSE;
        }
    }
    else
    {
        if (no_return_count > 2)
        {
            update_target_data = TRUE;
            scan_onto_target   = TRUE;
        }
    }

    if (fabs(normalize2(normalize(AUV_ST1000_bearing+psi)-target_bearing)) > 90.0)
    {
        if (TRUE && DISPLAYSCREEN)
            printf ("[Lost Target, Switching to Target Search Sonar Mode]\n");
        scan_onto_target   = TRUE;
        SONARSCANDIRECTION = -SONARSCANDIRECTION;
        SONARSCANMODE      = 4;
    }

    if (update_target_data)
    {
        if (valid_return_count > 0)
        {
            /* Compute a sonar range and bearing */
            target_bearing    = start_bearing;
            new_target_range  =
                range_accumulator / (double) valid_return_count;

            /* Determine location of target in world coordinates */
            target_x = x + cos_psi * AUV_ST1000_x_offset
                /* x position of sonar */
                - sin_psi * AUV_ST1000_y_offset;
            target_y = y + sin_psi * AUV_ST1000_x_offset
                /* y position of sonar */
                + cos_psi * AUV_ST1000_y_offset;
            target_x = target_x
                + cos (radians (target_bearing)) * new_target_range;
            target_y = target_y
                + sin (radians (target_bearing)) * new_target_range;

            /* Determine location of vehicle center relative to target */
            target_bearing = normalize (degrees (atan2z ((target_y - y),
                (target_x - x))));
            new_target_range = sqrt ((x - target_x) * (x - target_x) +
                (y - target_y) * (y - target_y));
        }
    }

```

```

        if (TRACE && DISPLAYSCREEN)
        {
            printf ("[Target X Location: %8.3lf]\n", target_x);
            printf ("[Target Y Location: %8.3lf]\n", target_y);
            printf ("[Target Bearing: %8.3lf]\n", target_bearing);
            printf ("[Target Range: %7.3lf]\n", target_range);
        }

        target_range      = new_target_range;
        new_target_update = TRUE;
        if (scan_onto_target)
        {
            range_accumulator      = 0.0;
            valid_return_count      = 0;
        }
        else
        {
            range_accumulator      = previous_range;
            valid_return_count      = 1;
        }
        no_return_count          = 0;
        SONARSCANDIRECTION       *= -1.0;
        time_last_target_update = t;

        fprintf (targetdatafile, "%lf %lf %lf %lf %lf %lf %lf\n",
            t, target_range, target_bearing, psi,
            target_range_command, target_bearing_command,
            psi_command_tgt);
        if (TRUE && DISPLAYSCREEN)
            printf ("[Target Update: %5.2lf %5.2lf]\n",
                target_range, target_bearing);
    }

    update_target_data      = FALSE;
}

if ((scan_onto_target) || (no_return_count == 0))
    ping_sonar (SONARSCANDIRECTION);
else ping_sonar (0);

break;

case (3): /* Target Tracking Mode */
    if (TRACE && DISPLAYSCREEN)
        printf ("[Sonar in target tracking mode]\n");

    /* See if it is time to reverse scan direction */
    if ((ST1000_range_kal > 0.0) &&
        ((ST1000_range_kal - previous_range) <= 5.0))
    {
        end_bearing = normalize (psi + AUV_ST1000_bearing);
        if (valid_return_count == 0)
            start_bearing = end_bearing;
        previous_range      = ST1000_range_kal;
        no_return_count      = 0;
        range_accumulator    += ST1000_range_kal;
        valid_return_count    += 1;
        ping_sonar (SONARSCANDIRECTION);
    }
    else
    {
        if (no_return_count > 3) /* scanned off end of target */
        {
            if (valid_return_count > 0)
            {
                /* Compute a sonar range and bearing */
            }
        }
    }
}

```

```

new_target_bearing = normalize (start_bearing +
                                normalize2 (end_bearing - start_bearing) / 2.0);
new_target_range =
range_accumulator / (double) valid_return_count;

/* Determine location of target in world coordinates */
target_x = x + cos_psi * AUV_ST1000_x_offset
            - sin_psi * AUV_ST1000_y_offset;
/* x position of sonar */
target_y = y + sin_psi * AUV_ST1000_x_offset
            + cos_psi * AUV_ST1000_y_offset;
/* y position of sonar */
target_x = target_x
            + cos (radians (new_target_bearing))
            * new_target_range;
target_y = target_y
            + sin (radians (new_target_bearing))
            * new_target_range;

/* Determine location of vehicle center relative to target */
target_bearing = normalize (degrees (atan2z ((target_y - y),
                                              (target_x - x))));
new_target_range = sqrt ((x - target_x) * (x - target_x) +
                          (y - target_y) * (y - target_y));

if (TRACE && DISPLAYSCREEN)
{
    printf ("[Target X Location: %8.3lf]\n", target_x);
    printf ("[Target Y Location: %8.3lf]\n", target_y);
    printf ("[Target Bearing: %8.3lf]\n", target_bearing);
    printf ("[Target Range: %7.3lf]\n", target_range);
}

target_range = new_target_range;
new_target_update = TRUE;
printf ("[Target Update: %5.2lf %5.2lf]\n",
        target_range, target_bearing);
fprintf (targetdatafile, "%lf %lf %lf %lf %lf %lf %lf\n",
        t, target_range, target_bearing, psi,
        target_range_command, target_bearing_command,
        psi_command_tgt);
time_last_target_update = t;
new_target_update = TRUE;
no_return_count = 0;
valid_return_count = 0;
range_accumulator = 0.0;
SONARSCANDIRECTION *= -1;
ping_sonar (SONARSCANDIRECTION);
}
else /* backtrack if scanned more than one step off target */
{
    ping_sonar (SONARSCANDIRECTION);
    no_return_count = 0;
}
else
{
    ping_sonar (0);
    ++no_return_count; /* try again before reversing scan */
}
}
break;

case (4): /* Find Target Based On Tactical Range and Bearing */
if (TRACE && DISPLAYSCREEN)
    printf ("[Sonar searching for target]\n");

```



```

previous_range      = 0.0;
time_last_target_update = 0.0;

if (TRACE && (ST1000_range_kal > 0.0))
{
    printf ("ST1000_range_kal: %5.11f\nSonar Bearing: %5.11f\n",
            ST1000_range_kal, normalize (AUV_ST1000_bearing + psi));
    printf ("Target Range: %5.11f\nTarget Bearing: %5.11f\n",
            target_range, target_bearing);
}

if (ST1000_range_kal > 0.0)
{
    /* Determine location of return in world coordinates */
    sonar_return_x = x + cos_psi * AUV_ST1000_x_offset
                    /* x position of sonar */
                    - sin_psi * AUV_ST1000_y_offset;
    sonar_return_y = y + sin_psi * AUV_ST1000_x_offset
                    /* y position of sonar */
                    + cos_psi * AUV_ST1000_y_offset;
    sonar_return_x += cos (radians (psi + AUV_ST1000_bearing))
                    * ST1000_range_kal;
    sonar_return_y += sin (radians (psi + AUV_ST1000_bearing))
                    * ST1000_range_kal;

    /* Determine location of vehicle center relative to target */
    new_target_bearing =
normalize (degrees (atan2z ((sonar_return_y - y), (sonar_return_x - x))));
    new_target_range = sqrt ( (x - sonar_return_x)
                              * (x - sonar_return_x)
                              + (y - sonar_return_y)
                              * (y - sonar_return_y));

    /* Determine if Target Located */
    if ((fabs (new_target_range - target_range) <= 5.0) &&
        (fabs (normalize2 (normalize (new_target_bearing) -
                                target_bearing)) <= 15.0))
    {
        if (TARGETEDGETRACK)
        {
            SONARSCANMODE = 2; /* use target edge scan */
            /* scan towards edge on side move is towards */
            SONARSCANDIRECTION =
dsign (normalize2 (target_bearing - target_bearing_command));
            if (TRUE && DISPLAYSCREEN)
            if (SONARSCANDIRECTION == 1)
                printf ("Sonar Scanning Right To Obtain Edge\n");
            else printf ("Sonar Scanning LEFT To Obtain Edge\n");
        }
        else
        {
            SONARSCANMODE = 3; /* use full target scan */
        }
        previous_range      = ST1000_range_kal;
        start_bearing       = normalize (psi+AUV_ST1000_bearing);
        no_return_count     = 0;
        scan_onto_target    = FALSE;
        range_accumulator   = ST1000_range_kal;
        valid_return_count  = 1;
        if (TRUE && DISPLAYSCREEN)
            printf ("[Target Located: %5.11f degrees, %5.11f feet]\n",
                    normalize (psi + AUV_ST1000_bearing), ST1000_range_kal);
    }
    ping_sonar (SONARSCANDIRECTION);
    break;
}

case (5): /* manual (tactical level) scan */

```

```

        if (TRACE && DISPLAYSCREEN)
        {
            printf ("[Sonar in manual scan mode]\n");
            printf ("[AUV_ST1000_bearing = %5.1lf]\n",AUV_ST1000_bearing);
            printf ("[Command ST1000_bearing = %5.1lf]\n",st1000_command);
        }

        commanded_bearing_error = normalize2(st1000_command-AUV_ST1000_bearing);

        if (commanded_bearing_error > SONARHEADINGSTEP / 2.0)
            SONARSCANDIRECTION = RIGHT;
        else if (commanded_bearing_error < -SONARHEADINGSTEP / 2.0)
            SONARSCANDIRECTION = LEFT;
        else SONARSCANDIRECTION = 0;
        ping_sonar (SONARSCANDIRECTION);
        break;

    default: /* Invalid Mode */
        if (DISPLAYSCREEN) printf ("[Invalid sonar scan mode! ***]\n");
        break;
}

if (TRACE && DISPLAYSCREEN)
    printf ("[End control_sonar]\n");
}
/*****

void open_virtual_world_socket () /* see os9sender.c for original code
*/
{
    if (LOCATIONLAB == FALSE) /* in water */
    {
        return;
    }

    if (TRACE && DISPLAYSCREEN)
        printf ("[start open_virtual_world_socket ()]\n");

/* Initialize communications blocks */
/* Initialize both client & server *****/

/* Signal handlers for termination to override net_open () and net_close
()*/
/* signal handlers. Otherwise you are unable to ^C kill this program.
*/

#ifdef os9
signal (SIGHUP, shutdown_virtual_world_socket);/* hangup */
signal (SIGINT, shutdown_virtual_world_socket);/* interrupt character */
/*
signal (SIGKILL, shutdown_virtual_world_socket);/* kill signal from Unix
*/
signal (SIGPIPE, shutdown_virtual_world_socket);/* broken pipe from other
host*/
signal (SIGTERM, shutdown_virtual_world_socket);/* software termination
*/
#endif

/* Initialize sender *****/

/* start by finding default/desired remote host to connect to */
{
    server_entity = gethostbyname (virtual_world_remote_host_name);
    if (server_entity == NULL)
    {

```



```

        virtual_world_socket_opened = FALSE;
        exit (1);
    }
    else if (TRACE && DISPLAYSCREEN)
    {
        printf ("[execution client connected to virtual world");
        printf (" server socket successfully]\n");
    }
    virtual_world_socket_opened = TRUE;

} /* end initialization */

socket_stream = socket_descriptor; /* client */

if (TRACE && DISPLAYSCREEN) /* print final info */
{
    printf("[open_virtual_world_socket CLIENT:  socket_descriptor = %d]\n",
           socket_descriptor);
    printf("[                               socket_accepted  = %d]\n",
           socket_accepted);
    printf("[                               socket_stream   = %d]\n",
           socket_stream);
}

if (TRACE && DISPLAYSCREEN)
    printf ("[finish open_virtual_world_socket ()]\n");

return;

}/* end open_virtual_world_socket () */

/*****
void shutdown_virtual_world_socket () /* see os9sender.c for original code
*/
{
    int kill_return_value;

    shutdown_signal_received = TRUE;

    if (LOCATIONLAB == FALSE) /* in water */
    {
        return;
    }

    if (virtual_world_socket_opened == FALSE)
    {
        if (TRACE && DISPLAYSCREEN)
        {
            printf ("[virtual_world_socket_opened FALSE,");
            printf (" shutdown_virtual_world_socket ignored]\n");
        }
        return;
    }
    if (TRACE && DISPLAYSCREEN)
        printf ("[shutdown_virtual_world_socket start ...]\n");

    /* No need to send a message to other side that bridge is going down,
    /*   since SIGPIPE signal trigger may shutdown server on other side */

    if (close (socket_stream) == -1)
    {
        if (TRACE && DISPLAYSCREEN)
            printf ("shutdown_virtual_world_socket close (socket_stream)
failed\n");

        /* shutdown () reference: "Using OS-9 Internet" manual p. 2-55 */

```

```

    if (shutdown (socket_stream, 2) == -1)
    {
        if (TRACE && DISPLAYSCREEN)
        {
            printf ("[shutdown_virtual_world_socket shutdown");
            printf (" (socket_stream, 2) failed]\n");
        }

        kill_return_value = kill (socket_stream, SIGKILL);

        if (TRACE && DISPLAYSCREEN)
        {
            printf ("[shutdown_virtual_world_socket kill (socket_stream,");
            printf (" SIGKILL) returned %d]\n", kill_return_value);
        }
    }
}
if (TRACE && DISPLAYSCREEN)
    printf ("[shutdown_virtual_world_socket return]\n");

return;
} /* end shutdown_virtual_world_socket () */

/*****
void send_buffer_to_virtual_world_socket () /* see os9sender.c for orig. code
*/
{
    bytes_left      = socket_length;
    bytes_written    = 0;
    ptr_index       = buffer; /* this global string is the data to be sent */

    if (LOCATIONLAB == FALSE) /* in water */
    {
        return;
    }

    if (virtual_world_socket_opened == FALSE)
    {
        if (TRACE && DISPLAYSCREEN)
        {
            printf ("[send_buffer_to_virtual_world_socket: ");
            printf ("virtual_world_socket_opened == FALSE, returning]\n");
        }
        return;
    }
    if (TRACE && DISPLAYSCREEN)
        printf ("[send_buffer_to_virtual_world_socket start ...]\n");

    while ((bytes_left > 0) && (bytes_written >= 0)) /* write loop *****/
    {
        bytes_sent = write (socket_stream, ptr_index, bytes_left);

        if (bytes_sent < 0) bytes_written = bytes_sent;
        else if (bytes_sent > 0)
        {
            bytes_left      -= bytes_sent;
            bytes_written    += bytes_sent;
            ptr_index        += bytes_sent;
        }

        if (LOCATIONLAB && TRACE && DISPLAYSCREEN)
        {
            printf ("[record_data send_telemetry_to_server loop");
            printf (" bytes sent = %d]\n", bytes_sent);
        }
    }
    if (bytes_written < 0)

```



```

        printf ("\n%s", buffer);
    }
    else if (DISPLAYSCREEN)                /* partial telemetry report */
    {
        if (count == 50)
        {
            printf("[t = %5.1f, voltages: %5.1f %5.1f]\n",
                    t, computer_voltage, motor_voltage);
            count = 0;
        }
        ++count;

        if (LOCATIONLAB && DISPLAYSCREEN)
        {
            printf ("sent telemetry to virtual world %5.2f ", t);
            printf ("(%5.1lf %5.1lf %5.1lf %5.1lf %5.1lf %5.1lf)\n",
                    x,y,z, phi,theta,psi);
        }
    }

    if (LOCATIONLAB)
    {
        get_string_from_virtual_world_socket ();    /* here it comes */
        if (REPLAY)
        {
            read_telemetry_string ();
            build_telemetry_string (buffer);
            build_telemetry_string (buffer_received);
        }
        else parse_telemetry_string (buffer_received);
    }

    if (TRACE && LOCATIONLAB && DISPLAYSCREEN)
    {
        printf ("-----\n");
    }

    if ( ((TACTICAL == FALSE) || (TACTICALPARSE)) && (auvdatafile != NULL)
        && (NOSCRIPT == FALSE))
        /* output data to telemetry file */
    {
        if (buffer_size == 0)                /* note that unmodified stream is saved */
            /* nothing was received, send auv_state */
            fprintf (auvdatafile, "%s", buffer);
        else
            /* feedback was received, send uvw_state */
            fprintf (auvdatafile, "%s", buffer_received);

        if (TRACE && DISPLAYSCREEN)
            printf("[printed to %s telemetry file]\n", AUVDATAFILENAME);
    }

    /* only send/print out every 10th telemetry entry to tactical level */
    /* due to serial port bandwidth limitations :-( */

    if ((TACTICAL) && TRACE && DISPLAYSCREEN)
        printf ("[sending data to tactical level]\n");

#ifdef os9
    /* -- NOT YET UPDATED TO CURRENT/OPERATIONAL MARCO AUV HARDWARE/SOFTWARE -- */
    /* writeln (serialpath, buffer, buffer_max); <<<<< */
    /* if (TACTICAL) write (serialpath, buffer, buffer_max); */

    if ((TACTICAL) && TRACE && DISPLAYSCREEN)
        printf ("[write buffer to tactical level serialpath OK]\n"); */
#endif
}

```



```

if (TACTICAL) send_buffer_to_tactical_socket (); /* telemetry */
if (auvtextfile != NULL)
/* output data to .auv text file */
{
    if (TRACE && DISPLAYSCREEN)
        printf ("[sending data to .auv text file]\n");

    fprintf (auvtextfile, "%s", buffer);
    if (buffer_size != 0) /*feedback was received, also send uvw_state */
        fprintf (auvtextfile, "%s", buffer_received);

    if (TRACE && DISPLAYSCREEN)
        printf ("[fprintf to .auv text file OK]\n");
}

telemetry_records_saved ++;

if (((buffer_index +1) % FILEBUFFERSIZE) == 0) buffer_index = 0;
else buffer_index ++;

/* need to copy buffer to buffer_array if caching telemetry <<<<<<<<<<<< */
if (TRACE && DISPLAYSCREEN) printf ("[buffer_index = %d]\n", buffer_index);

/* - - - - - */
/* test code to send data from file serial.d from wr2t1.c */
/* read characters from file, echo characters to screen, */
/* send characters to serialpath /t1 device */
/* while ((c[0] = getc(serialtestfile)) != EOF) */
{
    putc (c[0],stdout);
    writeln (serialpath, c, 1);
}
/*
/* - - - - - */

if (TRACE && DISPLAYSCREEN) printf ("[finish record_data ()]\n");

return;
}

/*****/

void execute_shutdown_script()
{
    static int phase = 1;
    static double time_next_phase;

    if (DISPLAYSCREEN && (phase == 1)) printf("[Shutdown Script Entered]\n");

    switch (phase)
    {
        case 1:
            if (TRUE && DISPLAYSCREEN) printf("[Starting Phase 1 of Shutdown Script]\n");
            THRUSTERCONTROL = TRUE;
            ROTATECONTROL = FALSE;
            LATERALCONTROL = FALSE;
            FOLLOWWAYPOINTMODE = FALSE;
            WAYPOINTCONTROL = FALSE;
            HOVERCONTROL = FALSE;
            GPSFIXINPROGRESS = FALSE;
            x_command = x;
    }
}

```

```

        y_command = y;
        z_command = -10;
        psi_command = psi;
        rpm = 0.0;
        port_rpm_command = 0.0;
        stbd_rpm_command = 0.0;
        time_next_phase = t + 20.0;
        phase = 2;
        break;
    case 2:
        if (t >= time_next_phase)
        {
            if (TRUE && DISPLAYSCREEN) printf("Starting Phase 2 of Shutdown Script\n");
            THRUSTERCONTROL = FALSE;
            time_next_phase = t + 1.0;
            phase = 3;
        }
        break;
    case 3:
        if (t >= time_next_phase)
        {
            if (TRUE && DISPLAYSCREEN) printf("Starting Phase 3 of Shutdown Script\n");
            LOOPFOREVER = FALSE;
            strcpy (buffer, "KILL");
            send_buffer_to_virtual_world_socket (); /* buffer msg sent */
            if (TRACE && DISPLAYSCREEN) printf ("\n[end_test set TRUE]\n");
            end_test = TRUE;

            if (NOSCRIPT == FALSE) fclose (auvscriptfile);
            auvscriptfilequit = TRUE;
            if (DISPLAYSCREEN)
                printf("\n[QUIT condition:(%s backup file) mission.script.backup,
                    file closed]\n",
                        AUVSCRIPTFILENAME);

            x_dot = 0.0;
            y_dot = 0.0;
            z_dot = 0.0;
            phi_dot = 0.0; /* degrees/sec */
            theta_dot = 0.0; /* degrees/sec */
            psi_dot = 0.0; /* degrees/sec */
            speed = 0.0;
            u = 0.0;
            v = 0.0;
            w = 0.0;
            p = 0.0; /* degrees/sec */
            q = 0.0; /* degrees/sec */
            r = 0.0; /* degrees/sec */
            delta_planes = 0.0; /* degrees */
            delta_rudder = 0.0; /* degrees */
            port_rpm = 0;
            stbd_rpm = 0;
            vertical_thruster_volts = 0.0;
            lateral_thruster_volts = 0.0;

            phase = 3;
        }
        break;
    default:
        break;
}
}
}

/*****

```

```

double read_computer_battery_voltage ()
{
    int    val;
    double voltage;

    val = get_adc1 (COMPUTER_VOLTAGE_CH);

    /* 3.03 Since a voltage divider Circuit by Approx 3 */
    voltage = (10.0/512.0) * ((double) val - 512.0) * 3.03;

    if (TRACE && DISPLAYSCREEN)
        printf ("read_computer_battery_voltage (): val      = %d\n", val);
    if (TRACE && DISPLAYSCREEN)
        printf ("read_computer_battery_voltage (): voltage = %f\n",
                voltage);

    return (voltage);
} /* end read_computer_battery_voltage () */

/*****/

double read_motor_gyro_battery_voltage ()
{
    int    val;
    double voltage;

    val = get_adc1 (MOTOR_GYRO_VOLTAGE_CH);
    /* 3.03 Since a voltage divider Circuit by Approx 3 */
    voltage = (10.0/512.0) * ((double) val - 512.0) * 3.03;

    if (TRACE && DISPLAYSCREEN)
        printf ("read_motor_gyro_battery_voltage (): val      = %d\n",val);
    if (TRACE && DISPLAYSCREEN)
        printf ("read_motor_gyro_battery_voltage (): voltage = %f\n",
                voltage);

    return (voltage);
} /* end read_motor_gyro_battery_voltage () */

/*****/

int leak_check ()
{
    int    i, adc_value [8], bow_adc_value, stern_adc_value;
    double bow_voltage, stern_voltage;

    LEAK = FALSE;

    for (i=0;i<8;++i)
    {
        adc_value [i] = get_adc1(i);
    }

    bow_adc_value = adc_value[5];
    bow_voltage   = (10.0/512.0)*(bow_adc_value - 512.0);

    if (TRACE && DISPLAYSCREEN)
        printf ("leak_check (): bow_voltage = %5.1f\n",bow_voltage);

    if (bow_voltage > 1.7)
    {
        if (DISPLAYSCREEN)
            printf("***** BOW LEAK DETECTED ***** bow_voltage = %f\n", bow_voltage);
        LEAK = TRUE;
    }
}

```

```

    }

    stern_adc_value = adc_value [6];
    if (TRACE && DISPLAYSCREEN)
        printf("stern_adc_value = %d\n",stern_adc_value);

    stern_voltage = (10.0/512.0)*(stern_adc_value - 512.0);

    if (TRACE && DISPLAYSCREEN)
        printf ("leak_check (): stern_voltage = %5.1f\n", stern_voltage);

    if (stern_voltage > 4.0) /* avoid spurious detections (originally 1.7) */
    {
        if (DISPLAYSCREEN)
            printf("***** STERN LEAK DETECTED ***** stern_voltage = %f\n",
                stern_voltage);
        LEAK = TRUE;
    }

    return (LEAK);

} /* end leak_check () */

/*****

/* Dive Tracker Functions Won't Compile on SGI */

#ifdef os9

int createdmod()
{
    mod_data *dat_struct;

    /* Create to data module */
    dt_dmod=CreateMod("DT2CL",sizeof(DT2CLMem),&dat_struct);
    if(dt_dmod == NULL) {
        if (DISPLAYSCREEN) printf("Data Module exists. Not created\n");
        exit(1);
    }
    dt_dmod->data_status=CL_R;
}

int CLReaddmod(r1_ptr,r2_ptr)
int *r1_ptr;
int *r2_ptr;
{

    static int r1,r2;

    if (dt_dmod->data_status==DT_W)
    {
        r1=dt_dmod->dtr.r1;
        r2=dt_dmod->dtr.r2;
        dt_dmod->data_status=CL_R;
        *r1_ptr=r1;
        *r2_ptr=r2;
        return(NEW_DATA);
    }
}

/
*****/
/*****change in code *****/
/***** Dave McClairen needs a negative value to signal that there is
no new data recorded by the dive tracker. *****/

/* *r1_ptr=-1;*/change in code from r1 to -1 to signal no update to tactical*/
/* *r2_ptr=-1;*/change in code from r2 to -1 to signal no update to tactical*/

```

```

    return(OLD_DATA);
}

void *AttachMod(str,data_struct)
char *str;
mod_data **data_struct;
{
    /* Trying to link to the Data Module */
    if ( (*data_struct=modlink(str,ANY))== (mod_data *)-1 ) {
        return(NULL);
    }

    /* ... then prepares an auxiliar structure to point to the Data Module */
    return( (void *)((long)((mod_data *)*data_struct)
                    +(long)(*data_struct)->data_offset) );
}

int DettachMod(data_struct)
mod_data *data_struct;
{
    if ( munlink(data_struct)== (mod_data *)-1 ) {
        return(-1);
    }
    return(0);
}

void *CreateMod(str,size,data_struct)
char *str;
unsigned size;
mod_data **data_struct;
{
    /* Creates Data Module */
    if( (*data_struct=_mkdata_module
        (str,size, mkattrevs(MA_REENT | MA_GHOST,0x00),Perm_field)) ==
        (mod_data *)-1
        ){
        return(NULL);
    }

    /* ... then prepares an auxiliar structure to point to the Data Module */
    return( (void *)((long)((mod_data *)*data_struct)
                    +(long)(*data_struct)->data_offset) );
}

#endif
/*****

/* Mathematical Model for Estimating X and Y */

void XY_model_est(v_ls,v_rs,v_blt,v_slt,X_dot_c,Y_dot_c,update_vel)

    double v_ls,v_rs,v_blt,v_slt,X_dot_c,Y_dot_c;
    unsigned short update_vel;
{
    double alpa_x = 0.0025,
          alpa_y = 0.004,
          b_x = 1.33,
          b_y = 17.0;

    double M_x = (435.0 + 43.5) / 32.2,
          M_y = (435.0 + 348.0) / 32.2;

    double f_ls,f_rs,f_blt,f_slt,F_x,F_y;
    double u_ddot,v_ddot,r_ddot;

```

```

f_ls = alpa_x*(v_ls*v_ls)*dsign(v_ls);
f_rs = alpa_x*(v_rs*v_rs)*dsign(v_rs);

f_blt = alpa_y*(v_blt*v_blt)*dsign(v_blt);
f_slt = alpa_y*(v_slt*v_slt)*dsign(v_slt);

F_x = f_ls + f_rs;
F_y = f_blt + f_slt;

/* Use speed sensor OR mathematical model to estimate speed */
/* depending on previous speed and speed sensor value */
u_ddot = (F_x - b_x*u*fabs(u))/M_x;
if ((u > 0.2) && (speed >= 0.2))
{
    u = speed;
}
else if ((u < -0.2) && (speed >= 0.2))
{
    speed = -speed;
    u = speed;
}
else
{
    u = u + dt * (u_ddot);

    if (u > 0.24) u = 0.24;
    else if (u < -0.24) u = -0.24;

    speed = u;
}

v_ddot = (F_y - b_y*v*fabs(v))/M_y;
v = v + dt*(v_ddot);

if(!update_vel)
{
    u = x_dot*cos_psi + y_dot*sin_psi;
    v = -x_dot*sin_psi + y_dot*cos_psi;
}

/* modify state vector values based on dead reckoning */

x_dot = u*cos_psi - v*sin_psi + X_dot_c;
y_dot = u*sin_psi + v*cos_psi + Y_dot_c;

x += dt * x_dot;
y += dt * y_dot;

return;
}

/*****

/* Constant gain Kalman filter for depth */
void kalman_z(yk)

double yk;
{
    double xk1_0,xk1_1,xk1_2;
    double phi1[3][3],h[3],b[3],lk[3],res;

    /* a=[0 1 0;0 0 1;0 0 0]; phi1=expm(a*0.1); where dt = 0.1 */
    b[0] = 0.0;
    b[1] = 0.0;
    b[2] = 1.0;

```

```

/* phii = [1.0    0.1    0.005
           0.0    1.0    0.1
           0.0    0.0    1.0] */
phii[0][0] = 1.0;
phii[0][1] = 0.1;
phii[0][2] = 0.005;
phii[1][0] = 0.0;
phii[1][1] = 1.0;
phii[1][2] = 0.1;
phii[2][0] = 0.0;
phii[2][1] = 0.0;
phii[2][2] = 1.0;

/* h = [1 0 0]; */
h[0] = 1.0;
h[1] = 0.0;
h[2] = 0.0;

if(kal_init_z == 1)
{
    z_kal = yk;
    z_dot_kal = 0.0;
    z_ddot_kal = 0.0;

    /* xk1=xk; */
    xk1_0 = z_kal;
    xk1_1 = z_dot_kal;
    xk1_2 = z_ddot_kal;
    kal_init_z = FALSE;
}

/* set lk = const. Slow Filter */
lk[0] = 0.2544;
lk[1] = 0.3727;
lk[2] = 0.2731;

/* xk1(:,i)=phii*xk(:,i); */
xk1_0 = phii[0][0]*z_kal + phii[0][1]*z_dot_kal + phii[0][2]*z_ddot_kal;
xk1_1 = phii[1][0]*z_kal + phii[1][1]*z_dot_kal + phii[1][2]*z_ddot_kal;
xk1_2 = phii[2][0]*z_kal + phii[2][1]*z_dot_kal + phii[2][2]*z_ddot_kal;

res = yk - (h[0]*xk1_0 + h[1]*xk1_1 + h[2]*xk1_2);

/* Set res = 0.0 if larger than threshold */
if(fabs(res) > thres_z)
{
    res = 0.0;
}

z_kal = xk1_0 + lk[0]*res;
z_dot_kal = xk1_1 + lk[1]*res;
z_ddot_kal = xk1_2 + lk[2]*res;

return;
}

/*****

/* Constant gain Kalman filter for ST1000 range */
void kalman_sonar1000(yk)

double yk;
{
    double xk1_0,xk1_1,xk1_2;
    double phii[3][3],h[3],b[3],lk[3],res;

    /* a=[0 1 0;0 0 1;0 0 0]; phii=expm(a*0.1); where dt = 0.1 */
    b[0] = 0.0;

```

```

b[1] = 0.0;
b[2] = 1.0;

/* phii = [1.0    0.1    0.005
           0.0    1.0    0.1
           0.0    0.0    1.0] */
phii[0][0] = 1.0;
phii[0][1] = 0.1;
phii[0][2] = 0.005;
phii[1][0] = 0.0;
phii[1][1] = 1.0;
phii[1][2] = 0.1;
phii[2][0] = 0.0;
phii[2][1] = 0.0;
phii[2][2] = 1.0;

/* h = [1 0 0]; */
h[0] = 1.0;
h[1] = 0.0;
h[2] = 0.0;

if(reset_sonar_filter)
{
    ST1000_range_kal = yk;
    ST1000_range_kal_dot = 0.0;
    ST1000_range_kal_ddot = 0.0;

    /* xk1=xk; */
    xk1_0 = ST1000_range_kal;
    xk1_1 = ST1000_range_kal_dot;
    xk1_2 = ST1000_range_kal_ddot;
    reset_sonar_filter = FALSE;
}

/* set lk = const. Slow Filter */
lk[0] = 0.2544;
lk[1] = 0.3727;
lk[2] = 0.2731;

/* xk1(:,i)=phii*xk(:,i); */
xk1_0 = phii[0][0]*ST1000_range_kal
        + phii[0][1]*ST1000_range_kal_dot
        + phii[0][2]*ST1000_range_kal_ddot;
xk1_1 = phii[1][0]*ST1000_range_kal
        + phii[1][1]*ST1000_range_kal_dot
        + phii[1][2]*ST1000_range_kal_ddot;
xk1_2 = phii[2][0]*ST1000_range_kal
        + phii[2][1]*ST1000_range_kal_dot
        + phii[2][2]*ST1000_range_kal_ddot;

res = yk - (h[0]*xk1_0 + h[1]*xk1_1 + h[2]*xk1_2);

/* Set res = 0.0 if larger than threshold */
if(fabs(res) > 5.0)
{
    res = 0.0;
}

ST1000_range_kal = xk1_0 + lk[0]*res;
ST1000_range_kal_dot = xk1_1 + lk[1]*res;
ST1000_range_kal_ddot = xk1_2 + lk[2]*res;
}

/*****

/* Constant gain Kalman filter for ST725 range */
void kalman_sonar725(yk)

```



```

double yk;
{
    double xk1_0,xk1_1,xk1_2;
    double phii[3][3],h[3],b[3],lk[3],res;

    /* a=[0 1 0;0 0 1;0 0 0]; phii=expm(a*0.1); where dt = 0.1 */
    b[0] = 0.0;
    b[1] = 0.0;
    b[2] = 1.0;

    /* phii = [1.0      0.1      0.005
               0.0      1.0      0.1
               0.0      0.0      1.0] */
    phii[0][0] = 1.0;
    phii[0][1] = 0.1;
    phii[0][2] = 0.005;
    phii[1][0] = 0.0;
    phii[1][1] = 1.0;
    phii[1][2] = 0.1;
    phii[2][0] = 0.0;
    phii[2][1] = 0.0;
    phii[2][2] = 1.0;

    /* h = [1 0 0]; */
    h[0] = 1.0;
    h[1] = 0.0;
    h[2] = 0.0;

    if(NEWRECOVERYCOMMAND)
    {
        ST725_range_kal = yk;
        ST725_range_kal_dot = 0.0;
        ST725_range_kal_ddot = 0.0;

        /* xk1=xk; */
        xk1_0 = ST725_range_kal;
        xk1_1 = ST725_range_kal_dot;
        xk1_2 = ST725_range_kal_ddot;
    }

    /* set lk = const. Slow Filter */
    lk[0] = 0.2544;
    lk[1] = 0.3727;
    lk[2] = 0.2731;

    /* xk1(:,i)=phii*xk(:,i); */
    xk1_0 = phii[0][0]*ST725_range_kal
            + phii[0][1]*ST725_range_kal_dot
            + phii[0][2]*ST725_range_kal_ddot;
    xk1_1 = phii[1][0]*ST725_range_kal
            + phii[1][1]*ST725_range_kal_dot
            + phii[1][2]*ST725_range_kal_ddot;
    xk1_2 = phii[2][0]*ST725_range_kal
            + phii[2][1]*ST725_range_kal_dot
            + phii[2][2]*ST725_range_kal_ddot;

    res = yk - (h[0]*xk1_0 + h[1]*xk1_1 + h[2]*xk1_2);

    /* Set res = 0.0 if larger than threshold */
    if(fabs(res) > thres_z)
    {
        res = 0.0;
    }

    ST725_range_kal      = xk1_0 + lk[0]*res;
    ST725_range_kal_dot  = xk1_1 + lk[1]*res;
    ST725_range_kal_ddot = xk1_2 + lk[2]*res;
}

```

```

/*****
/*****
/*****
/* end of execution.c
/*****
/*****
/*****/

```

IV. TACTICAL LEVEL SOURCE CODE

This section contains source code for the recovery path planner/command generator. The recovery path planner/command generator is responsible for planning a safe path from an arbitrary AUV position into an arbitrarily located recovery tube and generating commands that will be executed at the execution level to follow the planned path. This file is compiled and linked with other files of the tactical level. All tactical level software is available online at

http://www.stl.nps.navy.mil/~brutzman/dissertation/software_reference.html.

Files are available individually or as a group in a .tar package.

```

/*****
*
*   Filename:   recovery.c
*
*   Purpose:    Planner for smooth motion around and into a recovery tube
*
*   Reference:  Advanced Robotics class notes, Dr. Yutaka Kanayama
*
*   Author:     Duane Davis
*
*   Date:       12 June 96
*
*   Language:   ANSI C
*
*   Execution:  tactical
*
*****/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define TRUE 1
#define FALSE 0
#define LEFT -1
#define RIGHT 1

#define DELTA_S 0.01

#define TUBE_LENGTH 10.0
#define TUBE_RADIUS 1.75
#define ARC_DISTANCE 6.0

/* COMMANDFILE also used in tactical.c */
#define COMMANDFILE "recovery_points.auv"

#define TUBEFILE "tube.dat"
#define DATAFILE "recovery.dat"

#define LONGSPACING 6.0
#define CLOSESPACING 3.0

/* Enumeration for regions relative to tube */
enum { FRONT = 1, FRONT_LEFT_CORNER, LEFT_SIDE, REAR_LEFT_CORNER, LEFT_REAR,
      RIGHT_REAR, REAR_RIGHT_CORNER, RIGHT_SIDE, FRONT_RIGHT_CORNER };

/* Local Data Types */

typedef struct Point_type
{
    double x;
    double y;
} Point;

typedef struct Transformation_type
{
    Point point;
    double theta;
} Transformation;

typedef struct Vehicle_state_type
{
    Transformation posture;
    double kappa;
} Vehicle_state;

```

```

/* File Scope Variables */

double a;    /* steering coefficients */
double b;
double c;

double station_spacing = CLOSESPACING;
int delayed_new_region_output;
int first_point_of_plan;

FILE * datafile_ptr;    /* output recovery path points to a file */
FILE * tubefile_ptr;    /* output tube points to a file */
FILE * commandfile_ptr; /* file containing AUV station commands */

/* Global Variables from Other Files */

/* From statevector.c */
extern double x;
extern double y;

/* External Function Prototypes */

/* from circle.c */

extern double normalize_rad (double radian_angle);
extern double normalize2 (double degree_angle);

extern double degrees (double radian_angle);
extern double radians (double degree_angle);

/** Local Function Prototypes **/

Transformation compose (const Transformation* q1, const Transformation* q2);
Transformation inverse (const Transformation* q);
Transformation circular_transform (double l, double alpha);
double direction (const Point* p1, const Point* p2);
int included_angle (double angle, double lower_bound, double upper_bound);
int normal_to_segment (const Point* p, const Point* seg_pt_1,
                      const Point* seg_pt_2);
double steering_coefficients (double sigma, double* a, double* b, double* c);
double distance_to_point (const Point* p1, const Point* p2);
double distance_to_line (const Point* p, const Transformation* line);
double distance_to_circle (const Point* p, const Vehicle_state* curve);
double distance_to_segment (const Point * p, const Point * seg_pt_1,
                           const Point * seg_pt_2);
double theta_desired_circle (const Point* p, const Vehicle_state* curve);
void step_towards_line (Vehicle_state* vehicle, const Transformation* line);
void step_towards_circle (Vehicle_state* vehicle, const Vehicle_state* curve);
Point translate_point (Point* point, double x_trans, double y_trans,
                      double rotation);

```

```

void compute_tube_corners (double x, double y, double theta, Point*
corner_array);

void compute_track_lines (const Point* corner_array,
                          Vehicle_state* region2_track,
                          Transformation* region3_track,
                          Vehicle_state* region4_track,
                          Transformation* region5_track,
                          Transformation* region6_track,
                          Vehicle_state* region7_track,
                          Transformation* region8_track,
                          Vehicle_state* region9_track);

int determine_start_region (Vehicle_state* vehicle, const Point*
corner_array);

int output_station_point (const Point* station_pt, const Point* corner_array,
                          int region, int *new_region, const Point*
target_point);

int recovery_plan_complete (const Vehicle_state* vehicle,
                           double tube_x, double tube_y);

/* Function Definitions */

void
recovery_planner (tube_x, tube_y, tube_theta)

    double tube_x, tube_y, tube_theta;

{
    double sigma = 1.0,
          track_length = 0.0;
    int    vehicle_region,
          new_region = TRUE,
          recovery_file_complete = FALSE,
          counter;
    Point  corner_array[4],
          tube_center,
          previous_pts[100]; /* memory of previous 1' of track */

    Transformation region1, region3, region5, region6, region8;

    Vehicle_state region2, region4, region7, region9,
                 vehicle;

    delayed_new_region_output = FALSE;
    first_point_of_plan      = TRUE;

    /* Determine locations of tube corners and paths to track during recovery */
    tube_center.x = tube_x;
    tube_center.y = tube_y;
    compute_tube_corners (tube_x, tube_y, radians (tube_theta), corner_array);
    compute_track_lines (corner_array, &region2, &region3, &region4, &region5,
                        &region6, &region7, &region8, &region9);
    region1.point.x = tube_x;
    region1.point.y = tube_y;
    region1.theta   = radians (tube_theta);

    /* Initial posture for planning vehicle at AUV location, pointed on track */
    vehicle.posture.point.x = x;
    vehicle.posture.point.y = y;
    vehicle.posture.theta   = atan2 (tube_y - y, tube_x - x);
    vehicle.kappa           = 0.0;
    for (counter = 0; counter < 100; counter++)
        previous_pts[counter] = vehicle.posture.point;

```

```

steering_coefficients (sigma, &a, &b, &c);
vehicle_region = determine_start_region (&vehicle, corner_array);

/* set up data and command files */
if (((datafile_ptr = fopen (DATAFILE, "w")) == (FILE *) 0) ||
    ((tubefile_ptr = fopen (TUBEFILE, "w")) == (FILE *) 0))
{
    printf ("Error Opening Recovery Plan or Tube Data File\n");
    return;
}

if ((commandfile_ptr = fopen (COMMANDFILE, "w")) == (FILE *) 0)
{
    printf ("Error Opening Recovery Plan Command File\n");
    fclose (datafile_ptr);
    return;
}

fprintf (tubefile_ptr, "%7.4lf %7.4lf\n", corner_array[0].x,
        corner_array[0].y);
fprintf (tubefile_ptr, "%7.4lf %7.4lf\n", corner_array[1].x,
        corner_array[1].y);
fprintf (tubefile_ptr, "%7.4lf %7.4lf\n", corner_array[2].x,
        corner_array[2].y);
fprintf (tubefile_ptr, "%7.4lf %7.4lf\n", corner_array[3].x,
        corner_array[3].y);
fclose (tubefile_ptr);

/* Output first station point for initial position */
recovery_file_complete = output_station_point (&(vehicle.posture.point),
        corner_array, vehicle_region, &new_region,
        &tube_center);

while (recovery_file_complete == FALSE)
{
    switch (vehicle_region)
    {
        case (FRONT):
            step_towards_line (&vehicle, &region1);
            if ((distance_to_point (&(vehicle.posture.point), &tube_center) <=
                (TUBE_LENGTH / 2.0 + 2.5)) &&
                (recovery_file_complete == FALSE))
            {
                recovery_file_complete =
                    output_station_point (&(vehicle.posture.point),
                        corner_array, vehicle_region,
                        &new_region, &tube_center);
                track_length = 0.0;
            }
            break;
        case (FRONT_LEFT_CORNER):
            step_towards_circle (&vehicle, &region2);
            if (normal_to_segment (&(vehicle.posture.point),
                &corner_array[3], &corner_array[0]))
            {
                vehicle_region = FRONT;
                new_region = TRUE;
            }
            break;
        case (LEFT_SIDE):
            step_towards_line (&vehicle, &region3);
            if (! normal_to_segment (&(vehicle.posture.point),
                &corner_array[0], &corner_array[1]))
            {
                vehicle_region = FRONT_LEFT_CORNER;
                station_spacing = CLOSESPACING;
                new_region = TRUE;
            }
    }
}

```

```

        break;
    case (REAR_LEFT_CORNER):
        step_towards_circle (&vehicle, &region4);
        if (normal_to_segment (&(vehicle.posture.point),
                               &corner_array[0], &corner_array[1]))
        {
            vehicle_region = LEFT_SIDE;
            station_spacing = LONGSPACING;
            new_region = TRUE;
        }
        break;
    case (LEFT_REAR):
        step_towards_line (&vehicle, &region5);
        if (! normal_to_segment (&(vehicle.posture.point),
                                &corner_array[1], &corner_array[2]))
        {
            vehicle_region = REAR_LEFT_CORNER;
            new_region = TRUE;
        }
        break;
    case (RIGHT_REAR):
        step_towards_line (&vehicle, &region6);
        if (! normal_to_segment (&(vehicle.posture.point),
                                &corner_array[1], &corner_array[2]))
        {
            vehicle_region = REAR_RIGHT_CORNER;
            new_region = TRUE;
        }
        break;
    case (REAR_RIGHT_CORNER):
        step_towards_circle (&vehicle, &region7);
        if (normal_to_segment (&(vehicle.posture.point),
                               &corner_array[2], &corner_array[3]))
        {
            vehicle_region = RIGHT_SIDE;
            station_spacing = LONGSPACING;
            new_region = TRUE;
        }
        break;
    case (RIGHT_SIDE):
        step_towards_line (&vehicle, &region8);
        if (! normal_to_segment (&(vehicle.posture.point),
                                &corner_array[2], &corner_array[3]))
        {
            vehicle_region = FRONT_RIGHT_CORNER;
            station_spacing = CLOSESPACING;
            new_region = TRUE;
        }
        break;
    case (FRONT_RIGHT_CORNER):
        step_towards_circle (&vehicle, &region9);
        if (normal_to_segment (&(vehicle.posture.point),
                               &corner_array[3], &corner_array[0]))
        {
            vehicle_region = FRONT;
            new_region = TRUE;
        }
        break;
    default: break;
}

fprintf (datafile_ptr, "%7.4lf %7.4lf\n",
        vehicle.posture.point.x, vehicle.posture.point.y);

track_length += DELTA_S;
if (((track_length >= station_spacing) || (new_region)) &&
    (recovery_file_complete == FALSE))
{

```



```

        /* Use previous point to ensure the proper corner is the one used */
        recovery_file_complete = output_station_point (&(previous_pts[49]),
            corner_array, vehicle_region, &new_region,
            &tube_center);
        track_length = 0.0;
    }

    for (counter = 1; counter < 100; counter++)
        previous_pts[counter] = previous_pts[counter-1];
    previous_pts[0] = vehicle.posture.point;
}

fclose (datafile_ptr);
fclose (commandfile_ptr);
}

/* Functions for working with local data types */

/* Compose two transformations and return the result */
Transformation
compose (q1, q2)

    const Transformation* q1, *q2;

{
    Transformation result;
    double
        cos_q1_theta = cos (q1->theta),
        sin_q1_theta = sin (q1->theta);

    result.point.x = q1->point.x + q2->point.x * cos_q1_theta
                    - q2->point.y * sin_q1_theta;
    result.point.y = q1->point.y + q2->point.x * sin_q1_theta
                    + q2->point.y * cos_q1_theta;
    result.theta = q1->theta + q2->theta;

    return (result);
}

/* Compute the inverse of a transformation and return the result */
Transformation
inverse (q)

    const Transformation* q;

{
    Transformation result;
    double
        cos_theta = cos (q->theta),
        sin_theta = sin (q->theta);

    result.point.x = -q->point.x * cos_theta - q->point.y * sin_theta;
    result.point.y = q->point.x * sin_theta - q->point.y * cos_theta;
    result.theta = -q->theta;

    return (result);
}

/* Compute a circular transformation for length l and deltaTheta alpha */
Transformation
circular_transform (l, alpha)

    double l, alpha;

{
    Transformation result;

```

```

        double          alphaSquared = alpha * alpha,
                        alphaFourth  = alphaSquared * alphaSquared,
                        alphaSixth   = alphaSquared * alphaSquared * alphaSquared;

        result.point.x = 1 * (1 - (alphaSquared / 6.0)
                               + (alphaFourth / 120.0)
                               - (alphaSixth / 5040.0));
        result.point.y = 1 * alpha *
                        (0.5 - (alphaSquared / 24.0)
                        + (alphaFourth / 720.0)
                        - (alphaSixth / 40320.0));

        result.theta    = alpha;

        return (result);
    }

    /* Compute the direction from p1 to p2 */
    double
    direction (p1, p2)

        const Point* p1, *p2;

    {
        return (atan2 (p2->y - p1->y, p2->x - p1->x));
    }

    /* Determine whether an angle is between two bounding angles */
    int
    included_angle (angle, lower_bound, upper_bound)

        double angle, lower_bound, upper_bound;

    {
        double lower_to_upper    = normalize_rad (upper_bound - lower_bound),
               lower_to_included = normalize_rad (angle - lower_bound);

        if ((lower_to_upper < 0.0)    &&
            (lower_to_included <= 0.0) &&
            (lower_to_included >= lower_to_upper))
            return (TRUE);

        if ((lower_to_upper > 0.0)    &&
            (lower_to_included >= 0.0) &&
            (lower_to_included <= lower_to_upper))
            return (TRUE);

        if (lower_to_upper == lower_to_included)
            return (TRUE);

        return (FALSE);
    }

    /* Determine whether a ray beginning at a point normal to a line through
    /* two points p1 and p2 will intersect the closed segment defined by p1 & p2 */
    int
    normal_to_segment (p, seg_pt_1, seg_pt_2)

        const Point *p, *seg_pt_1, *seg_pt_2;

    {
        double dir_p_to_p1 = direction (p, seg_pt_1), /* Direction from p to p1 */
               dir_p_to_p2 = direction (p, seg_pt_2), /* Direction from p to p2 */
               dir_p_to_line, /* Direction from p to line through p1 & p2 */
               dist_p_to_line; /* distance from p to line through p1 & p2 */
    }

```

```

Transformation line;

line.theta = direction (seg_pt_1, seg_pt_2);
line.point = *seg_pt_1;

dist_p_to_line = distance_to_line (p, &line);

if (dist_p_to_line > 0)
    dir_p_to_line = normalize_rad (line.theta - M_PI / 2.0);
else dir_p_to_line = normalize_rad (line.theta + M_PI / 2.0);

if (included_angle (dir_p_to_line, dir_p_to_p1, dir_p_to_p2))
    return (TRUE);

return (FALSE);
}

/* Compute the distance between two points p1 and p2 */
double
distance_to_point (p1, p2)

    const Point *p1, *p2;

{
    double x_dist = p1->x - p2->x,
          y_dist = p1->y - p2->y;

    return (sqrt (x_dist * x_dist + y_dist * y_dist));
}

/* Compute the signed distance from a point to a directed line */
double
distance_to_line (p, line)

    const Point* p;
    const Transformation* line;

{
    double x_min_x0 = p->x - line->point.x,
          y_min_y0 = p->y - line->point.y,
          cos_theta = cos (line->theta),
          sin_theta = sin (line->theta);

    return (-x_min_x0 * sin_theta + y_min_y0 * cos_theta);
}

/* Compute the signed distance from a point to a circular curve */
double
distance_to_circle (p, curve)

    const Point* p;
    const Vehicle_state* curve;

{
    double x_min_x0      = p->x - curve->posture.point.x,
          y_min_y0      = p->y - curve->posture.point.y,
          cos_theta0     = cos (curve->posture.theta),
          sin_theta0     = sin (curve->posture.theta),
          k_x_min_x0     = curve->kappa * x_min_x0,
          k_y_min_y0     = curve->kappa * y_min_y0,
          numerator_term1 = -x_min_x0 * (k_x_min_x0 + 2.0 * sin_theta0),
          numerator_term2 = y_min_y0 * (k_y_min_y0 - 2.0 * cos_theta0),
          numerator       = numerator_term1 - numerator_term2,
          denominator_term1 = (k_x_min_x0 + sin_theta0) *
                              (k_x_min_x0 + sin_theta0),

```

```

        denominator_term2 = (k_y_min_y0 - cos_theta0) *
                             (k_y_min_y0 - cos_theta0),
        denominator      = 1 + sqrt (denominator_term1 + denominator_term2);

    return (numerator / denominator);
}

/* Determine the unsigned distance of a point from a line segment */
double
distance_to_segment (p, seg_pt_1, seg_pt_2)

    const Point *p, *seg_pt_1, *seg_pt_2;
{
    Transformation segment_line;

    double dist_p_to_p1,          /* Distance from p to p1          */
           dist_p_to_p2,          /* Distance from p to p2          */
           dist_p_to_line,        /* Distance from p to line through p1 & p2 */
           dir_p_to_p1 = direction (p, seg_pt_1), /* Direction from p to p1 */
           dir_p_to_p2 = direction (p, seg_pt_2), /* Direction from p to p2 */
           dir_p_to_line;         /* Direction from p to line through p1 & p2 */

    segment_line.theta = direction (seg_pt_1, seg_pt_2);
    segment_line.point = *seg_pt_1;

    dist_p_to_p1 = distance_to_point (p, seg_pt_1);
    dist_p_to_p2 = distance_to_point (p, seg_pt_2);

    if (normal_to_segment (p, seg_pt_1, seg_pt_2))
        return (fabs (distance_to_line (p, &segment_line)));

    if (dist_p_to_p1 < dist_p_to_p2)
        return (dist_p_to_p1);

    return (dist_p_to_p2);
}

/* Compute the tangential direction of a curve at the image of a point */
double
theta_desired_circle (p, curve)

    const Point* p;
    const Vehicle_state* curve;
{
    double k_x_min_x0 = curve->kappa * (p->x - curve->posture.point.x),
           k_y_min_y0 = curve->kappa * (p->y - curve->posture.point.y),
           sin_theta = sin (curve->posture.theta),
           cos_theta = cos (curve->posture.theta);

    return (atan2 (sin_theta + k_x_min_x0, cos_theta - k_y_min_y0));
}

/* Compute the steering coefficients given sigma */
double
steering_coefficients (sigma, a, b, c)

    double sigma, *a, *b, *c;
{
    double k = 1.0 / sigma;

```

```

    *a = 3.0 * k;
    *b = 3.0 * k * k;
    *c = k * k * k;
}

/* Move a vehicle one increment towards a line */
void
step_towards_line (vehicle, line)

    Vehicle_state* vehicle;
    const Transformation* line;

{
    double distance    = distance_to_line (&(vehicle->posture.point), line),
          theta_error = normalize_rad (vehicle->posture.theta - line->theta),
          dk_ds       = -(a * vehicle->kappa + b * theta_error + c * distance),
          delta_theta  = dk_ds * DELTA_S;

    Transformation incremental_motion;

    /* Move straight towards the line until we are close */
    if (fabs (distance) > 4.0)
    {
        dk_ds       = 0.0;
        delta_theta = 0.0;
    }

    incremental_motion = circular_transform (DELTA_S, delta_theta);
    vehicle->posture = compose (&(vehicle->posture), &incremental_motion);
}

/* Move a vehicle one increment towards a circular curve */
void
step_towards_circle (vehicle, curve)

    Vehicle_state* vehicle;
    const Vehicle_state* curve;

{
    double distance    = distance_to_circle (&(vehicle->posture.point),
curve),
          theta_error = normalize_rad (vehicle->posture.theta
- theta_desired_circle (&(vehicle->posture.point),
curve)),
          kappa_error = vehicle->kappa - curve->kappa,
          dk_ds       = -(a * vehicle->kappa + b * theta_error + c * distance),
          delta_theta  = dk_ds * DELTA_S;

    Transformation incremental_motion;

    /* Move straight towards the tube until we are close */
    if (fabs (distance) > 4.0)
    {
        dk_ds       = 0.0;
        delta_theta = 0.0;
    }

    incremental_motion = circular_transform (DELTA_S, delta_theta);
    vehicle->posture = compose (&(vehicle->posture), &incremental_motion);
}

/* Compute a 2-dimensional rotation and translation of a point */

```

```

Point
translate_point (point, x_trans, y_trans, rotation)

    Point* point;
    double x_trans, y_trans, rotation;

{
    double cos_theta = cos (rotation),
          sin_theta = sin (rotation);

    Point result;

    result.x = x_trans + cos_theta * point->x - sin_theta * point->y;
    result.y = y_trans + sin_theta * point->x + cos_theta * point->y;

    return (result);
}

/* Compute the (x,y) coordinates of the corners of the tube */
void
compute_tube_corners (x, y, theta, corner_array)

    double x, y, theta;
    Point *corner_array;

{
    corner_array[0].x = -TUBE_LENGTH / 2.0;
    corner_array[0].y = TUBE_RADIUS;
    corner_array[1].x = TUBE_LENGTH / 2.0;
    corner_array[1].y = TUBE_RADIUS;
    corner_array[2].x = TUBE_LENGTH / 2.0;
    corner_array[2].y = -TUBE_RADIUS;
    corner_array[3].x = -TUBE_LENGTH / 2.0;
    corner_array[3].y = -TUBE_RADIUS;

    corner_array[0] = translate_point (&corner_array[0], x, y, theta);
    corner_array[1] = translate_point (&corner_array[1], x, y, theta);
    corner_array[2] = translate_point (&corner_array[2], x, y, theta);
    corner_array[3] = translate_point (&corner_array[3], x, y, theta);
}

/* Compute the linear and circular paths along which the vehicle will track */
void
compute_track_lines (corner_array,
                    region2_track, region3_track, region4_track, region5_track,
                    region6_track, region7_track, region8_track, region9_track)

    const Point* corner_array; /* Corners of Recovery tube */
    Vehicle_state* region2_track; /* Track around front left corner */
    Transformation* region3_track; /* Track along left side */
    Vehicle_state* region4_track; /* Track around rear left corner */
    Transformation* region5_track; /* Track along left rear */
    Transformation* region6_track; /* Track along right rear */
    Vehicle_state* region7_track; /* Track around right rear corner */
    Transformation* region8_track; /* Track along right side */
    Vehicle_state* region9_track; /* Track around front right corner */

{
    double arc_curvature = 1.0 / ARC_DISTANCE,
          tube_orientation = direction (&corner_array[0], &corner_array[1]),
          cos_orientation = cos (tube_orientation),
          sin_orientation = sin (tube_orientation);

    Point region_2_3_pt, /* common point on path for region 2 and region 3 */
          region_4_5_pt, /* common point on path for region 4 and region 5 */
          region_6_7_pt, /* common point on path for region 6 and region 7 */

```

```

    region_8_9_pt;    /* common point on path for region 8 and region 9 */

region_2_3_pt.x = 0.0;
region_2_3_pt.y = ARC_DISTANCE;
region_2_3_pt = translate_point (&region_2_3_pt, corner_array[0].x,
                                corner_array[0].y, tube_orientation);

region_4_5_pt.x = ARC_DISTANCE;
region_4_5_pt.y = 0.0;
region_4_5_pt = translate_point (&region_4_5_pt, corner_array[1].x,
                                corner_array[1].y, tube_orientation);

region_6_7_pt.x = ARC_DISTANCE;
region_6_7_pt.y = 0.0;
region_6_7_pt = translate_point (&region_6_7_pt, corner_array[2].x,
                                corner_array[2].y, tube_orientation);

region_8_9_pt.x = 0.0;
region_8_9_pt.y = -ARC_DISTANCE;
region_8_9_pt = translate_point (&region_8_9_pt, corner_array[3].x,
                                corner_array[3].y, tube_orientation);

/* Arc around the front left corner of the tube */
region2_track->posture.point = region_2_3_pt;
region2_track->posture.theta = normalize_rad (tube_orientation + M_PI);
region2_track->kappa          = arc_curvature;

/* Line parallel to the left side of the tube oriented back to front */
region3_track->point = region_2_3_pt;
region3_track->theta = normalize_rad (tube_orientation + M_PI);

/* Arc around the back left corner of the tube */
region4_track->posture.point = region_4_5_pt;
region4_track->posture.theta = normalize_rad (tube_orientation + M_PI/2.0);
region4_track->kappa          = arc_curvature;

/* Line parallel to the back of the tube oriented center to left */
region5_track->point = region_4_5_pt;
region5_track->theta = normalize_rad (tube_orientation + M_PI / 2.0);

/* Line parallel to the back of the tube oriented center to right */
region6_track->point = region_6_7_pt;
region6_track->theta = normalize_rad (tube_orientation - M_PI / 2.0);

/* Arc around the back right corner of the tube */
region7_track->posture.point = region_6_7_pt;
region7_track->posture.theta = normalize_rad (tube_orientation - M_PI/2.0);
region7_track->kappa          = -arc_curvature;

/* Line parallel to the right side of the tube oriented back to front */
region8_track->point = region_8_9_pt;
region8_track->theta = normalize_rad (tube_orientation + M_PI);

/* Arc around the front right corner of the tube */
region9_track->posture.point = region_8_9_pt;
region9_track->posture.theta = normalize_rad (tube_orientation + M_PI);
region9_track->kappa          = -arc_curvature;
}

/* Determine what part of the tube the AUV is closest to at the start */
/* i.e. right side, right back corner, back side, etc. */
int
determine_start_region (vehicle, corner_array)

Vehicle_state* vehicle;

```

```

const Point*   corner_array;

{
    /* Determine ranges from four corners and from four sides */
    double dist_corner_array[4],
           dist_segment_array[4];

    dist_corner_array[0] = distance_to_point (&(vehicle->posture.point),
&corner_array[0]);
    dist_corner_array[1] = distance_to_point (&(vehicle->posture.point),
&corner_array[1]);
    dist_corner_array[2] = distance_to_point (&(vehicle->posture.point),
&corner_array[2]);
    dist_corner_array[3] = distance_to_point (&(vehicle->posture.point),
&corner_array[3]);
    dist_segment_array[0] = distance_to_segment (&(vehicle->posture.point),
&corner_array[0],
&corner_array[1]);
    dist_segment_array[1] = distance_to_segment (&(vehicle->posture.point),
&corner_array[1],
&corner_array[2]);
    dist_segment_array[2] = distance_to_segment (&(vehicle->posture.point),
&corner_array[2],
&corner_array[3]);
    dist_segment_array[3] = distance_to_segment (&(vehicle->posture.point),
&corner_array[3],
&corner_array[0]);

    /* Determine what segment or corner is closest return appropriate region */
    /* Also determine if vehicle is inside or outside track, if it is inside */
    /* Turn it around so that it points away from the tube */

    if ((dist_segment_array[3] < dist_corner_array[0]) &&
        (dist_segment_array[3] < dist_corner_array[3]) &&
        (dist_segment_array[3] < dist_segment_array[1]))
        return (FRONT);

    if ((dist_segment_array[3] == dist_segment_array[0]) &&
        (dist_segment_array[3] == dist_corner_array[0]))
    {
        if (dist_corner_array[0] < ARC_DISTANCE)
            vehicle->posture.theta = normalize_rad (vehicle->posture.theta +M_PI);
        return (FRONT_LEFT_CORNER);
    }

    if ((dist_segment_array[0] < dist_corner_array[0]) &&
        (dist_segment_array[0] < dist_corner_array[1]) &&
        (dist_segment_array[0] < dist_segment_array[2]))
    {
        if (dist_segment_array[0] < ARC_DISTANCE)
            vehicle->posture.theta = normalize_rad (vehicle->posture.theta +M_PI);
        station_spacing = LONGSPACING;
        return (LEFT_SIDE);
    }

    if ((dist_segment_array[0] == dist_segment_array[1]) &&
        (dist_segment_array[0] == dist_corner_array[1]))
    {
        if (dist_corner_array[1] < ARC_DISTANCE)
            vehicle->posture.theta = normalize_rad (vehicle->posture.theta +M_PI);
        return (REAR_LEFT_CORNER);
    }

    if ((dist_segment_array[1] < dist_corner_array[1]) &&
        (dist_segment_array[1] < dist_corner_array[2]) &&
        (dist_corner_array[1] < dist_corner_array[2]))
    {
        if (dist_segment_array[1] < ARC_DISTANCE)

```



```

        vehicle->posture.theta = normalize_rad (vehicle->posture.theta +M_PI);
        return (LEFT_REAR);
    }

    if ((dist_segment_array[1] < dist_corner_array[1]) &&
        (dist_segment_array[1] < dist_corner_array[2]) &&
        (dist_corner_array[2] <= dist_corner_array[1]))
    {
        if (dist_segment_array[1] < ARC_DISTANCE)
            vehicle->posture.theta = normalize_rad (vehicle->posture.theta +M_PI);
        return (RIGHT_REAR);
    }

    if ((dist_segment_array[2] == dist_segment_array[1]) &&
        (dist_segment_array[2] == dist_corner_array[2]))
    {
        if (dist_corner_array[2] < ARC_DISTANCE)
            vehicle->posture.theta = normalize_rad (vehicle->posture.theta +M_PI);
        return (REAR_RIGHT_CORNER);
    }

    if ((dist_segment_array[2] < dist_corner_array[2]) &&
        (dist_segment_array[2] < dist_corner_array[3]))
    {
        if (dist_segment_array[2] < ARC_DISTANCE)
            vehicle->posture.theta = normalize_rad (vehicle->posture.theta +M_PI);
        station_spacing = LONGSPACING;
        return (RIGHT_SIDE);
    }

    if ((dist_segment_array[2] == dist_segment_array[3]) &&
        (dist_segment_array[2] == dist_corner_array[3]))
    {
        if (dist_corner_array[3] < ARC_DISTANCE)
            vehicle->posture.theta = normalize_rad (vehicle->posture.theta +M_PI);
        return (FRONT_RIGHT_CORNER);
    }

    return (0);
}

/* Output an appropriate command for the AUV station point relative to tube */
int
output_station_point(station_pt,corner_array,region,new_region,final_point)
const Point *station_pt, *corner_array;
int region, *new_region;
const Point *final_point;
{
    static int region_1_station_pt = 1;

    double tube_heading = degrees (direction (&corner_array[0],
&corner_array[1]));

    char command_string [80] = "EDGE-STATION ",
        buffer          [25];

    double range,          /* range from vehicle to station keeping corner */
        bearing;          /* bearing from vehicle to station keeping corner */

    int sonar_scan_direction; /* scan direction to acquire correct corner */
    int return_value = 0;

    switch (region)
    {
        case (FRONT):

```

```

/* Final Recovery -- Sonars to side, move straight into tube */
if (distance_to_point (station_pt, final_point) <=
    (TUBE_LENGTH / 2.0 + 2.5))
    return_value = 1;

/* Coming from left side, use right corner for positioning */
if (region_1_station_pt == 4)
{
    sonar_scan_direction = LEFT;
    range = distance_to_point (station_pt, &corner_array[3]);
    bearing = degrees (direction (station_pt, &corner_array[3]));
}

/* Coming from right side, use left corner for positioning */
else
{
    sonar_scan_direction = RIGHT;
    range = distance_to_point (station_pt, &corner_array[0]);
    bearing = degrees (direction (station_pt, &corner_array[0]));
}
break;
case (FRONT_LEFT_CORNER):
    sonar_scan_direction = LEFT;
    if (*new_region)
    {
        range = distance_to_point (station_pt, &corner_array[0]);
        bearing = degrees (direction (station_pt, &corner_array[0]));
    }
    else
    {
        range = distance_to_point (station_pt, &corner_array[3]);
        bearing = degrees (direction (station_pt, &corner_array[3]));
    }
    region_1_station_pt = 4;
    break;
case (LEFT_SIDE):
    sonar_scan_direction = LEFT;
    range = distance_to_point (station_pt, &corner_array[0]);
    bearing = degrees (direction (station_pt, &corner_array[0]));
    break;
case (REAR_LEFT_CORNER):
    sonar_scan_direction = RIGHT;
    range = distance_to_point (station_pt, &corner_array[2]);
    bearing = degrees (direction (station_pt, &corner_array[2]));
    break;
case (LEFT_REAR):
    sonar_scan_direction = RIGHT;
    range = distance_to_point (station_pt, &corner_array[2]);
    bearing = degrees (direction (station_pt, &corner_array[2]));
    break;
case (RIGHT_REAR):
    sonar_scan_direction = LEFT;
    range = distance_to_point (station_pt, &corner_array[1]);
    bearing = degrees (direction (station_pt, &corner_array[1]));
    break;
case (REAR_RIGHT_CORNER):
    sonar_scan_direction = LEFT;
    range = distance_to_point (station_pt, &corner_array[1]);
    bearing = degrees (direction (station_pt, &corner_array[1]));
    break;
case (RIGHT_SIDE):
    sonar_scan_direction = RIGHT;
    range = distance_to_point (station_pt, &corner_array[3]);
    bearing = degrees (direction (station_pt, &corner_array[3]));
    break;
case (FRONT_RIGHT_CORNER):
    sonar_scan_direction = RIGHT;
    if (*new_region)

```

```

        {
            range = distance_to_point (station_pt, &corner_array[3]);
            bearing = degrees (direction (station_pt, &corner_array[3]));
        }
        else
        {
            range = distance_to_point (station_pt, &corner_array[0]);
            bearing = degrees (direction (station_pt, &corner_array[0]));
        }
        region_1_station_pt = 1;
        break;
    default:
        break;
}

/* First station in region, make sure AUV acquires correct corner */
if (((*new_region) &&
    (region != FRONT_RIGHT_CORNER) &&
    (region != FRONT_LEFT_CORNER)) ||
    (delayed_new_region_output) ||
    (first_point_of_plan))
{
    sprintf (buffer,"%lf %lf ",
        range, (bearing + sonar_scan_direction * 5.0));
    strcat (command_string, buffer);
    *new_region = FALSE;
    delayed_new_region_output = FALSE;
    first_point_of_plan = FALSE;
}
else if (*new_region)
{
    delayed_new_region_output = TRUE;
    *new_region = FALSE;
}

/* Print desired range and bearing to string */
sprintf (buffer,"%lf %lf", range, bearing);
strcat (command_string, buffer);

/* If in region 1, ensure AUV points in direction of tube */
if ((region == FRONT) ||
    ((region == FRONT_RIGHT_CORNER) &&
    (fabs (normalize2 (bearing - tube_heading)) < 45.0)) ||
    ((region == FRONT_LEFT_CORNER) &&
    (fabs (normalize2 (bearing - tube_heading)) < 45.0)))
{
    sprintf (buffer," %lf", tube_heading);
    strcat (command_string, buffer);
}

/* Print AUV command to file */
fprintf (commandfile_ptr,"%s\n",command_string);

if (return_value == 1)
{
    sprintf (command_string,"ENTER-TUBE %lf %lf",
        distance_to_point (station_pt, final_point),
        degrees (direction (&corner_array[0], &corner_array[1])));
    fprintf (commandfile_ptr,"%s\n",command_string);
}

return (return_value);
}

```

```

/* Determine when the recovery path plan has been completed */
int
recovery_plan_complete (vehicle, tube_x, tube_y)

    const Vehicle_state* vehicle;
    double tube_x, tube_y;

{
    double x_dist  = vehicle->posture.point.x - tube_x,
          y_dist  = vehicle->posture.point.y - tube_y,
          distance = sqrt (x_dist * x_dist + y_dist * y_dist);

    if (distance < DELTA_S)
        return (TRUE);

    return (FALSE);
}

/* End of File recovery.c */

```

V. MISSION GENERATION EXPERT SYSTEM SOURCE CODE

This section contains source code for the mission planning expert system. Source consists of three files: `mission_expert.pl`, `mission_cpp.c` and `mission_pl.c`. The file `mission_expert.pl` is written in Prolog and requires Quintus Prolog version 3.2 and XPCE for X-windows version 4.1 (Prowindows). The files `mission_cpp.c` and `mission_pl.c` are written in C and can be compiled on any platform. These files do however require script files (`controller_pl.script`, `controller_standalone_pl.script`, and `controller_cpp.script`) that are available online. These source and script files are available online individually or as part of the *Phoenix* AUV software package in .tar format at http://www.stl.nps.navy.mil/~brutzman/dissertation/software_reference.html

```

% File   : mission_expert.pl
% Author : Duane Davis, Brad Leonhardt
%
% Project: Pheonix AUV Strategic Level Mission Generation Expert System
%
% Purpose: Mission planning and generation expert system for the Phoenix
%          autonomous underwater vehicle
%
% System : Quintus Prolog 3.2 with pwxpce 3.0 (Prowindows)
%
% Use     : ai4/usr/work3/auv/strategic> newprowin
%          ?- [mission_expert].
%          ?- go.
%
% Date    : 16 May 96
%
% NOTE: This system only works on ai4. To run from another
% machine, you must telnet to ai4 and open a remote xterm.

:- ensure_loaded(library(math)),
   ensure_loaded(vehicle_info).
:- unknown(A, fail).
:- dynamic phase/5, start_phase/1, phase_list/1, current_phase/1,
   complete_successor/1, entry_mode/1, pathobject/1, abort_successor/1,
   x_scale/1, x_zero/1, y_scale/1, y_zero/1, searchpoint/1, goal_order/1,
   start_state/1, me_steps/1, next_phase/1, last_point/2,
   phase_summary_list/1, tube_data/4.

go :- quit, abolish(phase/5), abolish(start_phase/1), abolish(phase_list/1),
      abolish(entry_mode/1), asserta(phase_list([])), initial_menu.

```

```

% File   : control.pl
%
% Author : Duane Davis, Brad Leonhardt
%
% Project: Pheonix AUV Strategic Level Mission Generation Expert System
%
% Purpose: This file contains routines for drawing the initial system
%          menu, controlling the overall flow of the program, and
%          executing the phase by phase mission generation facility.
%
% System : Quintus Prolog 3.2 with pwxpce 3.0 (Prowindows)
%
% Use     : ai4/usr/work3/auv/strategic> newprowin
%          ?- [mission_expert].
%          ?- go.
%
% Date    : 26 April 96

```

```

initial_menu :- make_main_menu, make_phase_menu_labels.

/* make_main_menu creates a dialog box with choices for creating, modifying,
/* and deleting phases, and for getting means end help to plan a mission.
/* A chart of the operating area is also displayed with point and click
/* capability for entering navigation points */
make_main_menu :- destroy_start_window_objects, fail.
make_main_menu :-
    new(@start_dialog, dialog('Options')),

```

```

new(Start_menu,menu('Available Operations ',choice)),
send(Start_menu,layout,vertical),
new(Quit,button('Quit',message(@prolog,quit))),
new(Make,button('Generate Mission Code',message(@prolog,spec_complete))),
new(@file_name,text_item('Output File Name ','')),
new(Maps,menu('Available Charts ',cycle)),
send(Maps,layout,vertical),
construct_menu(Maps,['Moss Landing','Test Tank','NPS Pool'],get_map),
new(Label,label), send(Label,selection,
    'Coordinates from upper left corner.'),
new(P, path), assertz(pathobject(P)), send(P,pen,2),
new(@path_x1,text_item('Chart X ',0)),
new(@path_y1,text_item('Chart Y ',0)),
new(Path_clear,button('Clear Path',message(@prolog,erase_paths))),
new(@picture,picture), send(@picture,size,size(400,800)),
new(@bmp,bitmap),
send(@picture,clear), send(@picture,display,@bmp),
erase_paths, send(@picture,display,@mouse_click_path),
send(@start_dialog,size,size(800,150)),
construct_menu(Start_menu,['Phase by Phase Generation',
    'Modify Current Mission',
    'Means Ends Generation','Create Initialization File'],initial_choice),
send(@start_dialog,append,Start_menu),
send(@file_name,right,Start_menu), send(Maps,right,@file_name),
send(Quit,below,Start_menu), send(Make,right,Quit),
send(Path_clear,right,Make),
send(@path_x1,right,Path_clear), send(@path_y1,right,@path_x1),
send(@picture,below,@start_dialog), send(@start_dialog,open),
get_map('Moss Landing'),
send(@picture,recogniser,
    handler(ms_left_down,message(@prolog,left_button_down,@arg1))),
send(@picture,recogniser,
    handler(ms_right_down,message(@prolog,right_button_down,@arg1))).

/* make_phase_menu creates a dialog box which is used for displaying
/* the names of phases as they are specified. Each phase is displayed
/* with its complete and abort successor phase names */
make_phase_menu_labels :-
    pad_to_30('SPECIFIED PHASES',Phase_label),
    pad_to_30('COMPLETE SUCCESSORS',Comp_label),
    pad_to_30('ABORT SUCCESSORS',Abort_label),
    new(@phases,label), send(@phases,selection,Phase_label),
    new(@c_succ,label), send(@c_succ,selection,Comp_label),
    new(@a_succ,label), send(@a_succ,selection,Abort_label).

/* Display Phase Menu Opens the Phase Summary Window and */
/* Fills it puts labels for all of the phases into it */
display_phase_menu([[header,Phases,Completes,Aborts] | Rest]) :-
    free(@phase_dialog),
    new(@phase_dialog,dialog('Phase Summary')),
    send(@phase_dialog,append,Phases),
    send(Completes,right,Phases),
    send(Aborts,right,Completes),
    abolish(phase_summary_list/1),
    asserta(phase_summary_list([[header,Phases,Completes,Aborts] | Rest])),
    fail.
display_phase_menu([[Name1,Phase1,Complete1,Abort1],
    [Name2,Phase2,Complete2,Abort2] | Rest]) :-
    new(Phase_label,label), send(Phase_label,selection,Phase2),
    new(Complete_label,label), send(Complete_label,selection,Complete2),
    new(Abort_label,label), send(Abort_label,selection,Abort2),
    send(Phase_label,below,Phase1),
    send(Complete_label,right,Phase_label),
    send(Abort_label,right,Complete_label),

```

```

    display_phase_menu([Name2,Phase_label,Complete_label,Abort_label]
    | Rest]).
display_phase_menu([_]) :- send(@phase_dialog,open).

/* Alternates between moss landing, test tank, and NPS pool */
/* maps depending on menu selection from main dialog box */
get_map(_) :- erase_paths, fail.
get_map('Moss Landing') :- send(@bmp,load,xymoss),
    send(@picture,display,@bmp), consult(moss_info).
get_map('Test Tank') :- send(@bmp,load,tank),
    send(@picture,display,@bmp), consult(tank_info).
get_map('NPS Pool') :- send(@bmp,load,nps_pool),
    send(@picture,display,@bmp), consult(pool_info).

/* Menus for Creation, Deletion, and Modification of Phases */

/* Phase by Phase Mission Creation Menu */
/* phase_generation_menu makes a dialog box to get the type of phase
/* that the user wishes to enter. Types of phases are:
/* Change Depth, Transit, Hover, GPS Fix, Rotate AUV Search,
/* Rotate Sonar Search, Change Course, and Wait */
phase_generation_menu :- reset_to_main_menu,
    destroy_phase_entry_objects, fail.
phase_generation_menu :-
    new(@type_dialog,dialog('Phase Type')),
    new(Menu,menu('Press button for next operation',choice)),
    send(Menu,layout,vertical),
    new(Cancel,button('Cancel',message(@prolog,reset_to_main_menu))),
    construct_menu(Menu,['Depth Change','Course Change','Transit','Hover',
        'GPS Fix','Rotate Sonar Search','Rotate AUV Search','Wait',
        'Recover in Tube','Modify Phase','Delete Phase'],phase_info),
    send(@type_dialog,append,Menu),
    send(Cancel,below,Menu), send(@type_dialog,open).

/* Get input specific to different phase types */
/* phase_info is used when creating or modifying phases to get
/* information specific to each type of phase (as well as
/* information general to all types) */
phase_info('Delete Phase') :- \+phase(_,_,_,_),
    invalid_option_report(no_phases).
phase_info('Delete Phase') :- delete_phase.
phase_info('Modify Phase') :- \+phase(_,_,_,_),
    invalid_option_report(no_phases).
phase_info('Modify Phase') :- modify_phase.
phase_info(_) :- destroy_phase_entry_objects, make_common_items, fail.
phase_info('Depth Change') :- asserta(current_phase('Depth Change')),
    new(@parameter_dialog,dialog('Depth Change Parameters')),
    max_vehicle_depth(MaxZ),
    new(@new_depth,slider('New Depth ',0,MaxZ,0)),
    send(@parameter_dialog,append,@phase_name),
    send(@new_depth,below,@phase_name),
    send(@time_out,below,@new_depth), display_common_items.
phase_info('Course Change') :- asserta(current_phase('Course Change')),
    new(@parameter_dialog,dialog('Course Change Parameters')),
    new(@heading,slider('New Course ',0,360,0)),
    send(@parameter_dialog,append,@phase_name),
    send(@heading,below,@phase_name),
    send(@time_out,below,@heading), display_common_items.
phase_info('Wait') :- asserta(current_phase('Wait')),
    new(@parameter_dialog,dialog('Wait Parameters')),
    send(@parameter_dialog,append,@phase_name),
    send(@time_out,below,@phase_name), display_common_items.
phase_info('Transit') :- asserta(current_phase('Transit')),
    new(@parameter_dialog,dialog('Transit Parameters')),
    send(@parameter_dialog,append,@phase_name),

```



```

        make_x_y_depth_items,
        display_x_y_depth_items,
        send(@time_out,below,@new_y), display_common_items.
phase_info('Hover') :- asserta(current_phase('Hover')),
    new(@parameter_dialog,dialog('Hover Parameters')),
    send(@parameter_dialog,append,@phase_name),
    new(@heading,slider('Heading ',0,360,0)),
    make_x_y_depth_items,
    display_x_y_depth_items,
    send(@heading,below,@new_y),
    send(@time_out,below,@heading), display_common_items.
phase_info('GPS Fix') :- asserta(current_phase('GPS Fix')),
    new(@parameter_dialog,dialog('GPS Fix Parameters')),
    send(@parameter_dialog,append,@phase_name),
    send(@time_out,below,@phase_name), display_common_items.
phase_info('Rotate Sonar Search') :-
    asserta(current_phase('Rotate Sonar Search')),
    new(@parameter_dialog,dialog('Rotate Sonar Search Parameters')),
    send(@parameter_dialog,append,@phase_name),
    make_x_y_depth_items, display_x_y_depth_items,
    send(@time_out,below,@new_y), display_common_items.
phase_info('Rotate AUV Search') :-
    asserta(current_phase('Rotate AUV Search')),
    new(@parameter_dialog,dialog('Rotate AUV Search Parameters')),
    send(@parameter_dialog,append,@phase_name),
    make_x_y_depth_items, display_x_y_depth_items,
    send(@time_out,below,@new_y), display_common_items.
phase_info('Recover in Tube') :- asserta(current_phase('Recover in Tube')),
    new(@parameter_dialog,dialog('Tube Recovery Parameters')),
    send(@parameter_dialog,append,@phase_name),
    make_x_y_depth_items,
    send(@new_x,name,'Tube X Position '),
    send(@new_y,name,'Tube Y Position '),
    send(@new_depth,name,'Tube Depth '),
    display_x_y_depth_items,
    new(@heading,slider('Tube Entry Heading ',0,360,0)),
    send(@heading,below,@new_y),
    send(@time_out,below,@heading), display_common_items.
phase_info(_) :- display_common_items.

/* make_common_items, display_common_items, make_x_y_depth_items, and
/* display_x_y_depth_items are used for creating and displaying buttons
/* and text items for information and operations common to multiple
/* types of phases */
make_common_items :- new(@phase_name,text_item('Phase Name ','')),
    new(@time_out,slider('Time Out ',0,500,500)),
    new(@done,button('Done',message(@prolog,assert_phase))),
    new(@reset,button('Reset Phase',
        message(@prolog,destroy_phase_entry_objects))).

display_common_items :- phase_abort_successor(Abort_successor),
    phase_complete_successor(Complete_successor),
    send(Complete_successor,below,@time_out),
    send(Abort_successor,right,Complete_successor),
    send(@done,below,Complete_successor), send(@reset,right,@done),
    send(@parameter_dialog,open).

make_x_y_depth_items :-
    max_vehicle_depth(MaxZ),
    new(@new_depth,slider('Depth ',0,MaxZ,0)),
    get(@path_x1,selection,X), get(@path_y1,selection,Y),
    op_area(Xmin,Ymin,Xmax,Ymax),
    new(@new_x,slider('X Position ',Xmin,Xmax,X)),
    new(@new_y,slider('Y Position ',Ymin,Ymax,Y)).

```

```

display_x_y_depth_items :- send(@new_depth,below,@phase_name),
    send(@new_x,below,@new_depth),
    send(@new_y,below,@new_x).

/* phase_abort_successor and phase_complete_successor create menus to enter
/* the phases that will follow the phase being specified or modified */
phase_abort_successor(Abort_successor) :-
    new(Abort_successor,menu('Phase Abort Successor ',choice)),
    send(Abort_successor,layout,vertical),
    phase_list(List),append(['Unspecified' | List],
        ['mission_complete','mission_abort'],List2),
    construct_menu(Abort_successor,List2,handle_abort_successor).

phase_complete_successor(Complete_successor) :-
    new(Complete_successor,menu('Phase Complete Successor ',choice)),
    send(Complete_successor,layout,vertical),
    phase_list(List),append(['Unspecified' | List],
        ['mission_complete','mission_abort'],List2),
    construct_menu(Complete_successor,List2,handle_complete_successor).

handle_abort_successor(_) :- retract(abort_successor(X)), fail.
handle_abort_successor('Unspecified') :- get_unspecified_phase(abort).
handle_abort_successor(Successor) :- asserta(abort_successor(Successor)).

handle_complete_successor(_) :- retract(complete_successor(X)), fail.
handle_complete_successor('Unspecified') :- get_unspecified_phase(complete).
handle_complete_successor(Successor) :-
    asserta(complete_successor(Successor)).

/* Modify a previously specified phase */
/* modify_phase provides a menu with the names of all specified phases
/* for the user to choose from */
modify_phase :- destroy_phase_entry_objects,
    abolish(entry_mode/1), asserta(entry_mode(modify)), fail.
modify_phase :- new(@modify_dialog,dialog('Phase Modification')),
    new(Menu,menu('Press button for phase to modify',choice)),
    send(Menu,layout,vertical),
    phase_list(Phases), construct_menu(Menu,Phases,modify_phase),
    new(Reset,button('Reset',message(@prolog,destroy_phase_entry_objects))),
    send(@modify_dialog,append,Menu),
    send(Reset,below,Menu), send(@modify_dialog,open).

modify_phase(Phase) :- phase(Phase,Type,Parameters,CSuccessor,ASuccessor),
    phase_info(Type), asserta(complete_successor(CSuccessor)),
    asserta(abort_successor(ASuccessor)), replace_parameters(Phase,Type,Parameters).

/* Replace parameters marshals all of the previously specified parameters
/* for a phase and places it in the appropriate places in the data entry
/* dialog when a phase is being modified so that the data does not have to
/* be entered from scratch */
replace_parameters(Name,_,_) :- send(@phase_name,selection,Name), fail.
replace_parameters(_, 'Course Change', [Course, Time_out]) :-
    send(@heading,selection,Course), send(@time_out,selection,Time_out).
replace_parameters(_, 'Depth Change', [Depth, Time_out]) :-
    send(@new_depth,selection,Depth), send(@time_out,selection,Time_out).
replace_parameters(_, 'Transit', [X, Y, Depth, Time_out]) :-
    send(@new_x,selection,X), send(@new_y,selection,Y),
    send(@new_depth,selection,Depth), send(@time_out,selection,Time_out).
replace_parameters(_, 'Hover', [X, Y, Depth, Heading, Time_out]) :-
    send(@new_x,selection,X), send(@new_y,selection,Y),
    send(@new_depth,selection,Depth),
    send(@heading,selection,Heading), send(@time_out,selection,Time_out).
replace_parameters(_, 'GPS Fix', [Time_out]) :-

```

```

    send(@time_out,selection,Time_out).
replace_parameters(_,'Rotate Sonar Search',[X, Y, Depth, Time_out]) :-
    send(@new_x,selection,X), send(@new_y,selection,Y),
    send(@new_depth,selection,Depth), send(@time_out,selection,Time_out).
replace_parameters(_,'Rotate AUV Search',[X, Y, Depth, Time_out]) :-
    send(@new_x,selection,X), send(@new_y,selection,Y),
    send(@new_depth,selection,Depth), send(@time_out,selection,Time_out).
replace_parameters(_,'Wait',[Time_out]) :-
    send(@time_out,selection,Time_out).

/* Delete a previously specified phase */
/* delete_phase provides a list of all specified phases for the user
/* to choose from */
delete_phase :- destroy_phase_entry_objects,
    abolish(entry_mode/1), asserta(entry_mode(delete)), fail.
delete_phase :- new(@delete_dialog,dialog('Phase Deletion')),
    new(Menu,menu('Press button for phase to delete ',choice)),
    send(Menu,layout,vertical),
    phase_list(Phases), construct_menu(Menu,Phases,delete_phase),
    new(Reset,button('Reset',message(@prolog,destroy_phase_entry_objects))),
    send(@delete_dialog,append,Menu),
    send(Reset,below,Menu), send(@delete_dialog,open).
delete_phase(Phase) :- retract(phase(Phase,_,_,_)),
    pad_to_30(Phase,Phase_label),
    phase_summary_list(OldPSList),
    delete([_,Phase_label,_],OldPSList,NewPSList),
    display_phase_menu(NewPSList),
    retract(phase_list(PList)), delete(Phase,PList,NewPList),
    asserta(phase_list(NewPList)), destroy_phase_entry_objects.

% Append new point to most recent path when left button goes up;
% first path object will be the most recent
left_button_down(Event) :-
    extend_mouse_click_path(Event,X,Y),
    update_position_menus(X,Y).

right_button_down(Event) :-
    extend_mouse_click_path(Event,X,Y),
    update_transit_point_menus(X,Y).

extend_mouse_click_path(Event,Xrounded,Yrounded) :-
    get(Event,x,XPointer), get(Event,y,YPointer),
    x_zero(X_zero),x_scale(X_scale),
    Xcorrect is (((XPointer - X_zero) / (X_scale)) + 0.5),
    y_zero(Y_zero),y_scale(Y_scale),
    Ycorrect is (((YPointer - Y_zero) / (Y_scale)) + 0.5),
    floor(Xcorrect,Xrounded), floor(Ycorrect,Yrounded),
    send(@path_x1, selection, Yrounded),
    send(@path_y1, selection, Xrounded),
    send(@mouse_click_path,append,point(XPointer,YPointer)).

update_position_menus(X,Y) :-
    object(@new_x), send(@new_x, selection, Y),
    send(@new_y, selection, X), fail.
update_position_menus(X,Y) :-
    object(@trans_x), send(@trans_x, selection, Y),
    send(@trans_y, selection, X).
update_position_menus(X,Y).

update_transit_point_menus(X,Y) :-
    object(@trans_x), send(@trans_x, selection, Y),
    send(@trans_y, selection, X).
update_transit_point_menus(X,Y).

```

```

/* If a successor phase has not been specified its name is entered here */
/* The user is asked to enter the name of the unspecified phase. In
/* order for code to be generated by the program, the named phase must
/* be specified later */
get_unspecified_phase(_) :- destroy_unspecified_phase_entry_objects, fail.
get_unspecified_phase(abort) :- new(@ok1,button('OK',message(@pro-
log,abort_ok))),
    fail.
get_unspecified_phase(complete) :-
    new(@ok1,button('OK',message(@prolog,complete_ok))), fail.
get_unspecified_phase(_) :- new(@name_entry,dialog('Unspecified Phase Name')),
    new(Label1,label),
    send(Label1,selection,'Please enter the intended name of the unspecified
phase'),
    new(Label2,label),
    send(Label2,selection,
        'You will have to specify this phase before a mission can be generated'),
    new(@name,text_item('Name ','')),
    send(@name_entry,append,Label1), send(Label2,below,Label1),
    send(@name,below,Label2), send(@ok1,below,@name), send(@name_entry,open).

abort_ok :- get(@name,selection,Name), asserta(abort_successor(Name)),
    send(@name_entry,destroy).
complete_ok :- get(@name,selection,Name), asserta(complete_successor(Name)),
    send(@name_entry,destroy).

/* Callbacks for menus and pushbuttons */
initial_choice('Modify Current Mission') :- \+phase(_,_,_,_,_),
    invalid_option_report(no_phases).
initial_choice('Modify Current Mission') :- \+phase_summary_list(X),
    invalid_option_report(no_me_modify).
initial_choice(_) :- reset_to_main_menu, fail.
initial_choice('Create Initialization File') :- initialization_menu.
initial_choice('Modify Current Mission') :- phase_generation_menu.
initial_choice(_) :- abolish(phase/5), abolish(start_phase/1), fail.
initial_choice('Means Ends Generation') :- destroy_phase_summary_menu,
    means_end_menu.
initial_choice('Phase by Phase Generation') :- destroy_phase_summary_menu,
    display_phase_menu([[header,@phases,@c_succ,@a_succ]]),
    phase_generation_menu.

% File    : meansend.pl
%
% Author  : Neil Rowe
% Modified by: Duane Davis
%
% Project: Pheonix AUV Strategic Level Mission Generation Expert System
%
% Purpose: This file contains routines for conducting means ends analysis.
%          Operations, preconditions, and postconditions are used to
%          calculate a path from a start state to a desired goal state
%          Operations, pre and postconditions have been created for
%          various Pheonix AUV actions and capabilities.
%
% System  : Quintus Prolog 3.2 with pwxpce 3.0 (Prowindows)
%
% Use     : ai4/usr/work3/auv/strategic> newprowin
%          ?- [mission_expert].
%          ?- go.
%
% Date    : 29 April 96

```

```

means_ends(State,Goal,[],State) :- difference(Goal,State,[], !).
means_ends(State,Goal,Oplist,Goalstate) :- difference(Goal,State,D),
    recommended(Dsub,Operator),
    subset(Dsub,D),
    precondition(Operator,Prelist,NotPrelist),
    notprecondition(NotPrelist,State),
    means_ends(State,Prelist,Preoplist,Prestate),
    deletepostcondition(Operator,Deletepostlist),
    deleteitems(Deletepostlist,Prestate,Prestate2),
    addpostcondition(Operator,Addpostlist),
    union(Addpostlist,Prestate2,Postlist),
    means_ends(Postlist,Goal,Postoplist,Goalstate),
% nl,write('Completed -> '),write(Operator),nl,
    append(Preoplist,[Operator|Postoplist],Oplist).
means_ends(State,Goal,Oplist,Goalstate) :- difference(Goal,State,D),
    recommended(Dsub,Operator), subset(Dsub,D),
    \+precondition(Operator,Prelist,NotPrelist),
    write('Bug found: no preconditions found for operator '),
    write(Operator), nl, !, fail.
means_ends(State,Goal,Oplist,Goalstate) :- difference(Goal,State,D),
    recommended(Dsub,Operator), subset(Dsub,D),
    \+deletepostcondition(Operator,Deletepostlist),
    write('Bug found: no deletepostconditions found for operator '),
    write(Operator), nl, !, fail.
means_ends(State,Goal,Oplist,Goalstate) :- difference(Goal,State,D),
    recommended(Dsub,Operator), subset(Dsub,D),
    \+addpostcondition(Operator,Addpostlist),
    write('Bug found: no addpostconditions found for operator '),
    write(Operator), nl, !, fail.

```

```

difference([],S,[]).
difference([P|G],S,G2) :- singlemember(P,S), !, difference(G,S,G2).
difference([P|G],S,[P|G2]) :- difference(G,S,G2).

```

```

notprecondition([],State).
notprecondition([X|Tl],State) :- \+singlemember(X,State),
    notprecondition(Tl,State).

```

```

/* Recommended Operators to Achieve Goal States */
recommended([top_level_goal_done(X)], handle_top_level(X)).
recommended([position(X1,Y1,Z1), transit(X2,Y2,Z2)]),
    hover(X1,Y1,Z1,X2,Y2,Z2)).
recommended([position(X,Y,Z)], hover(X,Y,Z)).
recommended([no_fix_reqd],gps_fix).
recommended([no_dive_wait_reqd],dive_and_wait).
recommended([in_tube(X,Y,Z,Theta)],recover(X,Y,Z,Theta)).

```

```

/* Pre and Post Conditions for Different Phase Types */

```

```

/* Top Level Goals, One Operator With Different Operands */
/* to define different goals and handle them accordingly */

```

```

/* Search a Position, transit via a location prior to search */
precondition(handle_top_level([searched,X1,Y1,Z1,via,X2,Y2,Z2]),
    [[position(X1,Y1,Z1), transit(X2,Y2,Z2)], no_fix_reqd,
    no_dive_wait_reqd],[in_tube(_,_,_)])).
deletepostcondition(handle_top_level([searched,X1,Y1,Z1,via,X2,Y2,Z2]),
    [[position(X1,Y1,Z1),transit(X2,Y2,Z2)],no_fix_reqd])).
addpostcondition(handle_top_level([searched,X1,Y1,Z1,via,X2,Y2,Z2]),
    [top_level_goal_done([searched,X1,Y1,Z1,via,X2,Y2,Z2]),
    position(X1,Y1,Z1),fix_reqd])).

```

```

/* Search a Position, transit straight to the position */
precondition(handle_top_level([searched,X,Y,Z]),
              [position(X,Y,Z),no_fix_reqd,no_dive_wait_reqd],
              [in_tube(_,_,_)]) .
deletepostcondition(handle_top_level([searched,X,Y,Z]),
                    [no_fix_reqd]) .
addpostcondition(handle_top_level([searched,X,Y,Z]),
                 [top_level_goal_done([searched,X,Y,Z]),fix_reqd]) .

/* Lower Level Goals, accomplish high level goals preconditions */

/* Transit to point (X1,Y1,Z1) via point (X2,Y2,Z2) and hover */
precondition(hover(X1,Y1,Z1,X2,Y2,Z2),[no_fix_reqd,no_dive_wait_reqd],
              [in_tube(_,_,_)]) .
deletepostcondition(hover(X,Y,Z,X2,Y2,Z2),
                    [[position(OldX1,OldY1,OldZ1),transit(OldX2,OldY2,OldY3)],
                     position(OldX,OldY,OldZ)]) .
addpostcondition(hover(X,Y,Z,X2,Y2,Z2),[[position(X,Y,Z),transit(X2,Y2,Z2)]) .

/* Transit to point (X,Y,Z) and hover */
precondition(hover(X,Y,Z),[no_fix_reqd,no_dive_wait_reqd],[in_tube(_,_,_)]) .
deletepostcondition(hover(X,Y,Z),
                    [[position(OldX1,OldY1,OldZ1),transit(OldX2,OldY2,OldY3)],
                     position(OldX,OldY,OldZ)]) .
addpostcondition(hover(X,Y,Z),[position(X,Y,Z)]) .

/* Obtain a GPS fix */
precondition(gps_fix,[],[in_tube(_,_,_)]) .
deletepostcondition(gps_fix,[fix_reqd]) .
addpostcondition(gps_fix,[no_fix_reqd]) .

/* Dive to 2 feet and wait 30 seconds to initialize */
precondition(dive_and_wait,[],[in_tube(_,_,_)]) .
deletepostcondition(dive_and_wait,[]) .
addpostcondition(dive_and_wait,[no_dive_wait_reqd]) .

/* Recover in a torpedo tube */
precondition(recover(X,Y,Z,Theta),[],[in_tube(_,_,_)]) .
deletepostcondition(recover(X,Y,Z,Theta),[]) .
addpostcondition(recover(X,Y,Z,Theta),[in_tube(X,Y,Z,Theta)]) .

```

```

% File   : mehelp.pl
%
% Author : Duane Davis, Brad Leonhardt
%
% Project: Pheonix AUV Strategic Level Mission Generation Expert System
%
% Purpose: This file contains routines for specifying start state and
%          goal state conditions for an AUV mission. Means ends analysis
%          is used to compute a series of phases from the start state to the
%          goal state.
%
% System : Quintus Prolog 3.2 with pwxpce 3.0 (Prowindows)
%
% Use     : ai4/usr/work3/auv/strategic> newprowin
%          ?- [mission_expert].
%          ?- go.
%
% Date    : 29 April 96

```

```

/* means_end_menu brings up a menu that the user uses to
/* input the starting point of the vehicle, the recovery point
/* of the vehicle, and any points that the user wants searched.
/* Means ends analysis is used to compute a series of phases
/* that can be specified to accomplish the mission */
means_end_menu :- reset_to_main_menu, abolish(searchpoint/1), fail.
means_end_menu :- new(@medialog,dialog('Means End Help')),
    start_position(X,Y,Z),max_vehicle_depth(Zmax),
    op_area(Xmin,Ymin,Xmax,Ymax),
    new(Label1,label), send(Label1,selection,'Vehicle Initial Position'),
    new(@startx,slider('Launch X Position ',Xmin,Xmax,X)),
    new(@starty,slider('Launch Y Position ',Ymin,Ymax,Y)),
    new(@startz,slider('Launch Depth ',0,Zmax,Z)),
    new(Label2,label),send(Label2,selection,'Vehicle Recovery Position'),
    new(@endx,slider('Recovery X Position ',Xmin,Xmax,X)),
    new(@endy,slider('Recovery Y Position ',Ymin,Ymax,Y)),
    new(@endz,slider('Recovery Depth ',0,Zmax,Z)),
    new(Search_point,button('Enter Search Point',
        message(@prolog,enter_point))),
    new(Clear_points,button('Clear Search Points',
        message(@prolog,clear_search_points))),
    new(Generate,button('Generate Phase Sequence',
        message(@prolog,gen_phase_sequence))),
    new(Cancel,button('Cancel',message(@prolog,reset_to_main_menu))),
    new(Tube_recovery,button('Recover In Tube',
        message(@prolog,enter_tube_data))),
    new(Clear_tube,button('Cancel Tube Recovery',
        message(@prolog,clear_tube_data))),
    send(@medialog,append,Label1), send(@startx,below,Label1),
    send(@starty,below,@startx), send(@startz,below,@starty),
    send(@startz,below,@starty), send(Label2,below,@startz),
    send(@endx,below,Label2), send(@endy,below,@endx),
    send(@endz,below,@endy), send(Search_point,below,@endz),
    send(Tube_recovery,right,Search_point),
    send(Clear_points,below,Search_point),
    send(Clear_tube,below,Tube_recovery),
    send(Generate,right,Tube_recovery), send(Cancel,right,Clear_tube),
    send(@medialog,open).

/* gen_phase_sequence marshals all of the data from the means_end_menu
/* and all of the asserted searchpoints and calls the means end analysis
/* routine to get a series of phases to accomplish the mission */
gen_phase_sequence :- abolish(start_state/1), abolish(goal_order/1),
    get(@startx,selection,SX),
    get(@starty,selection,SY),
    get(@startz,selection,SZ),
    get(@endx,selection,EX),
    get(@endy,selection,EY),
    get(@endz,selection,EZ),
    marshal_goal_data(Sub_Goal_List),
    append([position(EX,EY,EZ)],Sub_Goal_List,Goal_List),
    asserta(start_state([position(SX,SY,SZ),no_fix_reqd])),
    means_ends([position(SX,SY,SZ),no_fix_reqd],Goal_List,Steps,_),
    asserta(goal_order(Goal_List)), reset_for_me_solution,
    mesolution(Steps), assert_phases(Steps).

/* next_solution finds a new solution using means ends analysis */
next_solution :- start_state([position(SX,SY,SZ) | L]),
    make_reordered_goal_list(Goal_List),
    means_ends([position(SX,SY,SZ) | Transit] | L,Goal_List,Steps,_),
    asserta(goal_order(Goal_List)), reset_for_me_solution,
    mesolution(Steps), assert_phases(Steps).

```

```

/* enter_point is used to enter points that need to be searched
/* searchpoint facts are asserted and later marshalled for use
/* in the means end analysis */
enter_point :- destroy_search_point_and_tube_objects,
    new(@pointdialog,dialog('Search Point Data')),
    new(Search_label,label),
    send(Search_label,selection,'Search Point Location'),
    make_x_y_depth_items,
    new(Take,button('Store Point',message(@prolog,store_point))),
    new(Cancel,button('Cancel',
        message(@prolog,destroy_search_point_and_tube_objects))),
    new(Trans_label,label),
    send(Trans_label,selection,'Transit Point Location:'),
    get(@path_x1,selection,X), get(@path_y1,selection,Y),
    op_area(Xmin,Ymin,Xmax,Ymax),
    new(@trans_x,slider('X Position ',Xmin,Xmax,X)),
    new(@trans_y,slider('Y Position ',Ymin,Ymax,Y)),
    send(@pointdialog,append,Search_label),
    send(@new_x,below,Search_label), send(@new_y,below,@new_x),
    send(@new_depth,below,@new_y), send(Trans_label,below,@new_depth),
    send(@trans_x,below,Trans_label), send(@trans_y,below,@trans_x),
    send(Take,below,@trans_y),
    send(Cancel,right,Take), send(@pointdialog,open).

store_point :- destroy_error_objects,
    invalid_point(Error), !, invalid_point_report(Error).
store_point :- get(@new_x,selection,X),
    get(@new_y,selection,Y),
    get(@new_depth,selection,Z),
    get(@trans_x,selection,TX),
    get(@trans_y,selection,TY),
    same(X,TX),same(Y,TY),
    asserta(searchpoint(top_level_goal_done([searched,X,Y,Z]))),
    destroy_search_point_and_tube_objects.
store_point :- get(@new_x,selection,X),
    get(@new_y,selection,Y),
    get(@new_depth,selection,Z),
    get(@trans_x,selection,TX),
    get(@trans_y,selection,TY),
    asserta(searchpoint(top_level_goal_done([searched,X,Y,Z,via,TX,TY,Z]))),
    destroy_search_point_and_tube_objects.

/* enter tube data is used to enter the location of */
/* the tube into which the auv is to recover */
enter_tube_data :- destroy_search_point_and_tube_objects,
    new(@tubedialog,dialog('Recovery Tube Data')),
    make_x_y_depth_items,
    send(@new_x,name,'Tube X Position '),
    send(@new_y,name,'Tube Y Position '),
    send(@new_depth,name,'Tube Depth '),
    new(@heading,slider('Tube Entry Heading',0,360,0)),
    new(Take,button('Store Data',message(@prolog,store_tube_data))),
    new(Cancel,button('Cancel',
        message(@prolog,destroy_search_point_and_tube_objects))),
    get(@path_x1,selection,X), get(@path_y1,selection,Y),
    send(@tubedialog,append,@new_x),
    send(@new_y,below,@new_x), send(@new_depth,below,@new_y),
    send(@heading,below,@new_depth), send(Take,below,@heading),
    send(Cancel,right,Take), send(@tubedialog,open).

/* Store Tube Data marshals all the recovery tube location */
/* data and asserts a fact with the tube location for use later */
store_tube_data :- destroy_error_objects,
    invalid_point(Error), !, invalid_point_report(Error).
store_tube_data :- get(@new_x,selection,X),
    get(@new_y,selection,Y),

```



```

    get(@new_depth,selection,Z),
    get(@heading,selection,Theta),
    abolish(tube_data/4), asserta(tube_data(X,Y,Z,Theta)),
    perpendicular_point(X,Y,Theta,Xapproach,Yapproach),
    send(@endx,selection,Xapproach),
    send(@endy,selection,Yapproach),
    send(@endz,selection,Z),
    destroy_search_point_and_tube_objects.

/* mesolution takes the sequence of phases derived through
/* means ends analysis and displays them in a dialog box */
mesolution([LastStep]) :- object(LastStep), !,
    new(Cancel,button('Cancel',message(@prolog,destroy_me_soln_objects))),
    new(NextSoln,button('Next Solution',message(@prolog,next_solution))),
    send(Cancel,below,LastStep), send(NextSoln,right,Cancel),
    send(@solutiondialog,open).
mesolution(Steps) :- \+object(@solutiondialog),
    abolish(me_steps/1), asserta(me_steps(Steps)),
    new(@solutiondialog,dialog('Means End Solution')),
    new(Label,label), send(Label,selection,'Computed Phase Sequence:'),
    new(Solutionlabel,label),
    send(@solutiondialog,append,Label), send(Solutionlabel,below,Label),
    mesolution([Solutionlabel | Steps]).
mesolution([Previous, Current | Rest]) :- me_step_label(Current,Label),
    new(Step_label,label), send(Step_label,selection,Label),
    send(Step_label,below,Previous), mesolution([Step_label | Rest]).

make_reordered_goal_list(Goal_List) :-
    goal_order([position(X,Y,Z) | Used_List]),
    same_members(Sub_Goal_List,Used_List),
    \+goal_order([position(_,_,_) | Sub_Goal_List]),
    same([position(X,Y,Z) | Sub_Goal_List],Goal_List).
make_reordered_goal_list(Goal_List) :-
    goal_order(Goal_List), abolish(goal_order/1).

/* me_step_label matches phase commands derived through means ends
/* analysis with more user friendly text strings */
me_step_label(handle_top_level([searched,X,Y,Z,via,_,_,_]),Label) :-
    list_concat(['Conduct a sonar search at position X=',X,' Y=',Y,
    ' Depth=',Z,'.'],Label).
me_step_label(handle_top_level([searched,X,Y,Z]),Label) :-
    list_concat(['Conduct a sonar search at position X=',X,' Y=',Y,
    ' Depth=',Z,'.'],Label).
me_step_label(sonar_search(X,Y,Z,_,_,_),Label) :-
    list_concat(['Conduct a sonar search at position X=',X,' Y=',Y,
    ' Depth=',Z,'.'],Label).
me_step_label(hover(X,Y,Z),Label) :-
    list_concat(['Hover at position X=',X,' Y=',Y,' Depth=',Z,'.'],Label).
me_step_label(hover(X,Y,Z,XT,YT,ZT),Label) :-
    list_concat(['Transit to Position X =',XT,' Y=',YT,' Depth=',ZT,
    ', then hover at position X=',X,' Y=',Y,' Depth=',Z,'.'],Label).
me_step_label(gps_fix,'Obtain a GPS fix').
me_step_label(dive_and_wait,'Dive to 2 feet and wait 30 seconds').
me_step_label(recover(X,Y,Z,Theta),Label) :-
    list_concat(['Recover in Tube at Position X = ',X,' Y = ',Y,
    ' Depth = ',Z,' heading ',Theta,' degrees'],Label).

/* Use the Results from Means Ends Analysis to assert phase facts */
assert_phases([]) :- !.
assert_phases([dive_and_wait | Rest]) :- next_phase(Current),
    get_next_phase([dive_and_wait | Rest],Next),
    assertz(phase(Current,'Depth Change',[2,100],Next,'mission_abort')),
    get_next_phase(Rest,Next2),
    assertz(phase(Next,'Wait',[30],Next2,Next2)),

```

```

    assert_phases(Rest).
assert_phases([hover(X,Y,Z) | Rest]) :- next_phase(Current),
    get_next_phase(Rest,Next), get_fail_next(Current,Rest,FailNext),
    assertz(phase(Current,'Hover',[X,Y,Z,0,500],Next,FailNext)),
    extend_me_route(X,Y), assert_phases(Rest).
assert_phases([hover(X,Y,Z,XT,YT,ZT) | Rest]) :- next_phase(Current),
    CurrentPlus1 is Current + 1,
    get_next_phase([hover(X,Y,Z,XT,YT,ZT) | Rest],Next),
    assertz(phase(Current,'Transit',[XT,YT,ZT,500],Next,CurrentPlus1)),
    extend_me_route(XT,YT), assert_phases([hover(X,Y,Z) | Rest]).
assert_phases([gps_fix | Rest]) :- next_phase(Current),
    get_next_phase(Rest,Next),
    get_fail_next(Current,Rest,FailNext),
    assertz(phase(Current,'GPS Fix',[150],Next,FailNext)),
    assert_phases(Rest).
assert_phases([handle_top_level([searched,X,Y,Z,via,_,_,_]) | Rest]) :-
    next_phase(Current),
    get_next_phase(Rest,Next), get_fail_next(Current,Rest,FailNext),
    assertz(phase(Current,'Rotate Sonar Search',[X,Y,Z,250],Next,FailNext)),
    assert_phases(Rest).
assert_phases([handle_top_level([searched,X,Y,Z]) | Rest]) :-
    next_phase(Current),
    get_next_phase(Rest,Next), get_fail_next(Current,Rest,FailNext),
    assertz(phase(Current,'Rotate Sonar Search',[X,Y,Z,250],Next,FailNext)),
    assert_phases(Rest).
assert_phases([recover(X,Y,Z,Theta) | Rest]) :-
    next_phase(Current), get_next_phase(Rest,Next),
    assertz(phase(Current,'Recover in Tube',[X,Y,Z,Theta,500],
        Next,'mission_abort')), assert_phases(Rest).

/* next phase in list that is either a transit or hover */
get_next_transit_phase([], 'mission_abort') := !.
get_next_transit_phase([First | Rest], 1) :- same(First, hover(_,_,_,_,_)), !.
get_next_transit_phase([First | Rest], 1) :- same(First, hover(_,_,_)), !.
get_next_transit_phase([First | Rest], X) :- get_next_transit_phase(Rest, Y),
    number(Y), X is Y + 1, !.
get_next_transit_phase(_, 'mission_abort').

/* if a phase fails, the followon phase will be the next transit */
get_fail_next(Current, Rest, FailNext) :-
    get_next_transit_phase(Rest, JumpOnFail),
    number(JumpOnFail), FailNext is Current + JumpOnFail.
get_fail_next(Current, Rest, 'mission_abort').

/* next phase in the list */
get_next_phase([], 'mission_complete').
get_next_phase(L, Next) :- retract(next_phase(Current)), Next is Current + 1,
    asserta(next_phase(Next)).

extend_me_route(_, _) :- \+object(@me_route), new(@me_route, path),
    send(@me_route, pen, 2),
    get(@startx, selection, X), get(@starty, selection, Y),
    x_zero(X_zero), x_scale(X_scale), y_zero(Y_zero), y_scale(Y_scale),
    XDraw is ((Y * X_scale) + X_zero - 0.5 * X_scale), floor(XDraw, XDraw2),
    YDraw is ((X * Y_scale) + Y_zero - 0.5 * Y_scale), floor(YDraw, YDraw2),
    send(@me_route, append, point(XDraw2, YDraw2)),
    send(@picture, display, @me_route), fail.
extend_me_route(X, Y) :-
    x_zero(X_zero), x_scale(X_scale), y_zero(Y_zero), y_scale(Y_scale),
    XDraw is ((Y * X_scale) + X_zero - 0.5 * X_scale), floor(XDraw, XDraw2),
    YDraw is ((X * Y_scale) + Y_zero - 0.5 * Y_scale), floor(YDraw, YDraw2),
    send(@me_route, append, point(XDraw2, YDraw2)).

marshal_goal_data(Data_list) :- tube_data(X,Y,Z,Theta),
    bagof_searchpoints(Point_list),
    append([in_tube(X,Y,Z,Theta)], Point_list, Data_list).
marshal_goal_data(Data_list) :- bagof_searchpoints(Data_list).

```

```
bagof_searchpoints(Point_List) :- bagof(X,searchpoint(X),Point_List), !.
bagof_searchpoints([]).
```

```
% File   : generator.pl
%
% Author  : Duane Davis, Brad Leonhardt
%
% Project: Pheonix AUV Strategic Level Mission Generation Expert System
%
% Purpose: This file contains routines for asserting phase facts (used to
%           generate the command_strings file), creating the command_strings
%           file, and generating C++ and/or Prolog Strategic Level Code
%
% System  : Quintus Prolog 3.2 with pwxpce 3.0 (Prowindows)
%
% Use     : ai4/usr/work3/auv/strategic> newprowin
%           ?- [mission_expert].
%           ?- go.
%
% Date    : 29 April 96
```

```
/* Callback when mission specification complete */
spec_complete :- \+phase(_,_,_,_), invalid_option_report(no_phases).
spec_complete :- destroy_error_objects, \+start_phase(X), !,
    new(@first_phase,dialog('Start Phase')),
    new(Phase_menu,menu('Select Desired First Phase: ',choice)),
    send(Phase_menu,layout,vertical),
    phase_list(Phase_list), construct_menu(Phase_menu,Phase_list,assert_start),
    send(@first_phase,append,Phase_menu), send(@first_phase,open).
spec_complete :- parse_mission.

assert_start(Start) :- abolish(start_phase/1), asserta(start_phase(Start)),
    send(@first_phase,destroy), parse_mission.
```

```
/* assert_phase asserts a fact of the form: */
/* phase(Name, Type, Parameter_list, Complete_successor, Abort_successor) */
/* for each phase specified by the user */
assert_phase :- invalid_phase(Error), !, invalid_phase_report(Error).
assert_phase :- abolish(entry_mode/1), get(@phase_name,selection,Phase_name),
    retract(phase(Phase_name,_,_,_)),
    retract(phase_list(PList)), delete(Phase_name,PList,NewPList),
    asserta(phase_list(NewPList)), fail.
assert_phase :- current_phase(Phase),
    phase_parameters(Phase,Parameters), get(@phase_name,selection,Phase_name),
    complete_successor(CSuccessor),abort_successor(ASuccessor),
    asserta(phase(Phase_name,Phase,Parameters,CSuccessor,ASuccessor)),
    retract(phase_list(Phase_list)), asserta(phase_list([Phase_name |
Phase_list])),
    pad_to_30(Phase_name,Name_string), pad_to_30(CSuccessor,CSucc_string),
    pad_to_30(ASuccessor,ASucc_string),
    phase_summary_list(OldPList),
    delete([_,Name_string,_,_],OldPList,OldPList2),
    append(OldPList2,[Phase,Name_string,CSucc_string,ASucc_string],NewPList),
    display_phase_menu(NewPList),
    destroy_phase_entry_objects.
```

```
/* phase_parameters marshals the parameters required for different phase types
```

```

*/
/* into a list for inclusion in the phase facts asserted by assert_phase */
phase_parameters('Depth Change',[Depth, Time_out]) :-
    get(@new_depth,selection,Depth),
    get(@time_out,selection,Time_out).
phase_parameters('Course Change',[Heading, Time_out]) :-
    get(@heading,selection,Heading),
    get(@time_out,selection,Time_out).
phase_parameters('Wait',[Time_out]) :-
    get(@time_out,selection,Time_out).
phase_parameters('Transit',[X, Y, Depth, Time_out]) :-
    get(@new_x,selection,X),
    get(@new_y,selection,Y),
    get(@new_depth,selection,Depth),
    get(@time_out,selection,Time_out).
phase_parameters('Hover',[X, Y, Depth, Heading, Time_out]) :-
    get(@new_x,selection,X),
    get(@new_y,selection,Y),
    get(@heading,selection,Heading),
    get(@new_depth,selection,Depth),
    get(@time_out,selection,Time_out).
phase_parameters('GPS Fix',[Time_out]) :-
    get(@time_out,selection,Time_out).
phase_parameters('Rotate Sonar Search',[X, Y, Depth, Time_out]) :-
    get(@new_x,selection,X),
    get(@new_y,selection,Y),
    get(@new_depth,selection,Depth),
    get(@time_out,selection,Time_out).
phase_parameters('Rotate AUV Search',[X, Y, Depth, Time_out]) :-
    get(@new_x,selection,X),
    get(@new_y,selection,Y),
    get(@new_depth,selection,Depth),
    get(@time_out,selection,Time_out).
phase_parameters('Recover in Tube',[X, Y, Depth, Heading, Time_out]) :-
    get(@new_x,selection,X),
    get(@new_y,selection,Y),
    get(@heading,selection,Heading),
    get(@new_depth,selection,Depth),
    get(@time_out,selection,Time_out).

```

```

/* Check mission for errors and generate code if mission valid*/
parse_mission :- destroy_error_objects, fail.
parse_mission :- mission_error(Phase,loop), !,
invalid_mission_report(Phase,loop).
parse_mission :- phase(Phase,_,_,_),
    setof(Error,mission_error(Phase,Error),Error_list), !,
    destroy_error_objects, phase_error_report(Phase,Error_list),
    mission_error(_,unreachable), abolish(start_phase/1).
parse_mission :- tell(command_strings), fail.
parse_mission :- get(@file_name,selection,File_name),
    start_phase(Start_phase),
    list_concat(['file_name ',File_name,' ',Start_phase],Command),
    write(Command),nl, fail.
parse_mission :- phase(Phase,_,_,_), generate_code(Phase), fail.
parse_mission :- told, code_dialog.

```

```

/* Generate code writes a series of strings to a file called "mission_strings"
/* These strings are used by the C program to generate code in whatever
/* language is required. The current version generates Prolog code. */
generate_code(Phase) :-
    phase(Phase,'Recover in Tube',[X,Y,Depth,Theta,Time_out],CSucc,ASucc),
    list_concat(['tube_recovery ',Phase,' ',CSucc,' ','withdraw ',
        Time_out,' ',X,' ',Y,' ',Depth,' ',Theta],Command1),
    write(Command1),nl,

```

```

    approach_point(X,Y,Theta,Xout,Yout),
    list_concat(['hoverpoint withdraw ',ASucc,' ',ASucc,' 250 ',
        Xout,' ',Yout,' ',Depth,' ',Theta],Command2),
    write(Command2),nl.
generate_code(Phase) :- phase(Phase,'Depth
Change',[Depth,Time_out],CSucc,ASucc),
    list_concat(['depth_change ',Phase,' ',CSucc,' ',ASucc,' ',
        Time_out,' ',Depth],Command),
    write(Command),nl.
generate_code(Phase) :- phase(Phase,'Course Change',[Head-
ing,Time_out],CSucc,ASucc),
    list_concat(['course ',Phase,' ',CSucc,' ',ASucc,' ',
        Time_out,' ',Heading],Command),
    write(Command),nl.
generate_code(Phase) :- phase(Phase,'Wait',[Time_out],CSucc,ASucc),
    list_concat(['wait ',Phase,' ',CSucc,' ',ASucc,' ',Time_out],Command),
    write(Command),nl.
generate_code(Phase) :-
    phase(Phase,'Transit',[X,Y,Depth,Time_out],CSucc,ASucc),
    list_concat(['waypoint ',Phase,' ',CSucc,' ',ASucc,' ',
        Time_out,' ',X,' ',Y,' ',Depth],Command),
    write(Command),nl.
generate_code(Phase) :- phase(Phase,'Hover',[X,Y,Depth,Head-
ing,Time_out],CSucc,ASucc),
    list_concat(['hoverpoint ',Phase,' ',CSucc,' ',ASucc,' ',
        Time_out,' ',X,' ',Y,' ',Depth,' ',Heading],Command),
    write(Command),nl.
generate_code(Phase) :- phase(Phase,'GPS Fix',[Time_out],CSucc,ASucc),
    list_concat(['get_gps_fix ',Phase,' ',CSucc,' ',ASucc,' ',Time_out],
        Command), write(Command),nl.
generate_code(Phase) :-
    phase(Phase,'Rotate Sonar Search',[X,Y,Depth,Time_out],CSucc,ASucc),
    list_concat(['rotate_sonar_search ',Phase,' ',CSucc,' ',ASucc,' ',
        Time_out,' ',X,' ',Y,' ',Depth,' '],Command),
    write(Command),nl.
generate_code(Phase) :- phase(Phase,'Rotate AUV
Search',[X,Y,Depth,Time_out],CSucc,ASucc),
    list_concat(['sonar_search ',Phase,' ',CSucc,' ',ASucc,' ',
        Time_out,' ',X,' ',Y,' ',Depth,' '],Command),
    write(Command),nl.

code_dialog :- destroy_code_selection_objects, fail.
code_dialog :- new(@code_dialog,dialog('Language')),
    new(Label,label),
    send(Label,selection,'Parsing Complete, Choose Desired Output Language:'),
    new(Cpp,button('C++',message(@prolog,language,'C++'))),
    new(Prolog,button('Prolog',message(@prolog,language,'Prolog'))),
    new(Done,button('Cancel',message(@prolog,destroy_code_selection_objects))),
    send(@code_dialog,append,Label), send(Cpp,below,Label),
    send(Prolog,right,Cpp), send(Done,right,Prolog), send(@code_dialog,open).

language('C++') :- unix(shell('mission_cpp')), destroy_code_selection_objects.
language('Prolog') :- unix(shell('mission_pl')),
    destroy_code_selection_objects.

```

```

% File : initialization.pl
%
% Author : Duane Davis, Brad Leonhardt
%
% Project: Pheonix AUV Strategic Level Mission Generation Expert System
%
% Purpose: This file contains routines for specifying and creating the
%          initialization file for an AUV mission. The initialization
%          file contains mission specific information not required at the
%          strategic level such as dive tracker locations, initial vehicle

```

```

%           posture, and gyro error.
%
% System : Quintus Prolog 3.2 with pwxpce 3.0 (Prowindows)
%
% Use      : ai4/usr/work3/auv/strategic> newprowin
%           ?- [mission_expert].
%           ?- go.
%
% Date     : 29 April 96

```

```

initialization_menu :- reset_to_main_menu, \+object(@initialize_dialog),
    start_position(X,Y,Z),
    max_vehicle_depth(Zmax), op_area(Xmin,Ymin,Xmax,Ymax),
    new(@initialize_dialog,dialog('Initialization Parameters')),
    new(Posture_label,label),
    send(Posture_label,selection,'Initial Posture'),
    new(@posture_x,slider('X ',Xmin,Xmax,X)),
    new(@posture_y,slider('Y ',Xmin,Xmax,Y)),
    new(@posture_z,slider('Depth ',0,Zmax,Z)),
    new(@posture_phi,slider('Roll Angle ',0,360,0)),
    new(@posture_theta,slider('Pitch Angle ',0,360,0)),
    new(@posture_psi,slider('Heading ',0,360,0)),
    new(Blank1,label),
    new(DT1_label,label),
    send(DT1_label,selection,'Dive Tracker Unit 1 Location'),
    new(@dt1x,slider('X ',Xmin,Xmax,0)),
    new(@dt1y,slider('Y ',Ymin,Ymax,0)),
    new(@dt1z,slider('Depth ',0,Zmax,2)),
    new(Blank2,label),
    new(DT2_label,label),
    send(DT2_label,selection,'Dive Tracker Unit 2 Location'),
    new(@dt2x,slider('X ',Xmin,Xmax,0)),
    new(@dt2y,slider('Y ',Ymin,Ymax,0)),
    new(@dt2z,slider('Depth ',0,Zmax,2)),
    new(Blank3,label),
    new(@gyro_error,slider('Gyro Error ',0,360,0)),
    new(Done,button('Done',message(@prolog,make_init_file))),
    new(Cancel,button('Cancel',message(@prolog,reset_to_main_menu))),
    send(@initialize_dialog,append,Posture_label),
    send(@posture_x,below,Posture_label),
    send(@posture_y,below,@posture_x),
    send(@posture_z,below,@posture_y),
    send(@posture_phi,below,@posture_z),
    send(@posture_theta,below,@posture_phi),
    send(@posture_psi,below,@posture_theta),
    send(Blank1,below,@posture_psi),
    send(DT1_label,below,Blank1),
    send(@dt1x,below,DT1_label),
    send(@dt1y,below,@dt1x),
    send(@dt1z,below,@dt1y),
    send(Blank2,below,@dt1z),
    send(DT2_label,below,@dt1z),
    send(@dt2x,below,DT2_label),
    send(@dt2y,below,@dt2x),
    send(@dt2z,below,@dt2y),
    send(Blank3,below,@dt2z),
    send(@gyro_error,below,Blank3),
    send(Done,below,@gyro_error), send(Cancel,right,Done),
    send(@initialize_dialog,open).

```

```

make_init_file :- tell('initialization.script'),
    get(@posture_x,selection,PX),
    get(@posture_y,selection,PY),

```

```

get(@posture_z,selection,PZ),
get(@posture_phi,selection,PPhi),
get(@posture_theta,selection,PTheta),
get(@posture_psi,selection,PPsi),
list_concat(['POSTURE ',PX,' ',PY,' ',PZ,' ',PPhi,' ',PTheta,' ',PPsi],
Posture),
list_concat(['HEADING ',PPsi],Heading),
get(@dt1x,selection,Dt1X),
get(@dt1y,selection,Dt1Y),
get(@dt1z,selection,Dt1Z),
list_concat(['DIVE-TRACKER1 ',Dt1X,' ',Dt1Y,' ',Dt1Z],DT1),
get(@dt2x,selection,Dt2X),
get(@dt2y,selection,Dt2Y),
get(@dt2z,selection,Dt2Z),
list_concat(['DIVE-TRACKER2 ',Dt2X,' ',Dt2Y,' ',Dt2Z],DT2),
get(@gyro_error,selection,G_error),
list_concat(['GYRO-ERROR ',G_error],Gyro_Error),
write(Posture), nl,nl,
write(Heading), nl,nl,
write(DT1), nl,nl,
write(DT2), nl,nl,
write(Gyro_Error), nl,nl,
write('THRUSTERS-ON'), nl,nl,
told, reset_to_main_menu.
..

% File : errors.pl
%
% Author : Duane Davis, Brad Leonhardt
%
% Project: Pheonix AUV Strategic Level Mission Generation Expert System
%
% Purpose: This file contains rules for determining when invalid phases,
%          missions, or operations are attempted, and routines for
%          displaying error messages as appropriate.
%
% System : Quintus Prolog 3.2 with pwxpce 3.0 (Prowindows)
%
% Use : ai4/usr/work3/auv/strategic> newprowin
%      ?- [mission_expert].
%      ?- go.
%
% Date : 29 April 96

/* Rules for determining when a specified phase is invalid */
invalid_phase(no_name) :- get(@phase_name,selection,'').
invalid_phase(duplicate_phase) :- \+entry_mode(modify),
    get(@phase_name,selection,Name), phase(Name,_,_,_).
invalid_phase(no_successor) :- \+abort_successor(X).
invalid_phase(no_successor) :- abort_successor('').
invalid_phase(no_successor) :- \+complete_successor(X).
invalid_phase(no_successor) :- complete_successor('').
invalid_phase(non_number) :- object(@time_out),get(@time_out,selection,String),
    \+string_to_num(String,_).
invalid_phase(non_number) :- object(@new_x),get(@new_x,selection,String),
    \+string_to_num(String,_).
invalid_phase(non_number) :- object(@new_y),get(@new_y,selection,String),
    \+string_to_num(String,_).
invalid_phase(non_number) :- object(@new_depth),get(@new_depth,selection,String),
    \+string_to_num(String,_).
invalid_phase(non_number) :- object(@heading),get(@heading,selection,String),

```



```

        phase(Phase, 'Rotate Sonar Search', [X,Y,Z,_],_,_),
        \+phase(_, 'Hover',_,_,Phase), \+phase(_, 'Hover',_,_,Phase).
mission_error(Phase,displaced_search) :-
    phase(Phase, 'Rotate AUV Search', [X,Y,Z,_],_,_),
    phase(_,Type,Parameters,Phase,_),
    (\+same(Type, 'Hover')) ; \+same(Parameters, [X,Y,Z,_],_)).
mission_error(Phase,displaced_search) :-
    phase(Phase, 'Rotate AUV Search', [X,Y,Z,_],_,_),
    phase(_,Type,Parameters,_,Phase),
    (\+same(Type, 'Hover')) ; \+same(Parameters, [X,Y,Z,_],_)).
mission_error(Phase,displaced_search) :-
    phase(Phase, 'Rotate AUV Search', [X,Y,Z,_],_,_),
    \+phase(_, 'Hover',_,_,Phase), \+phase(_, 'Hover',_,_,Phase).

/* Rules for determining when a specified search or transit point is invalid */
/* Search and transit points are generated by the means ends facility */
invalid_point(non_number) :- object(@new_x),get(@new_x,selection,String),
    \+string_to_num(String,_).
invalid_point(non_number) :- object(@new_y),get(@new_y,selection,String),
    \+string_to_num(String,_).
invalid_point(non_number) :- object(@new_depth),
    get(@new_depth,selection,String),
    \+string_to_num(String,_).
invalid_point(non_number) :- object(@trans_x),get(@trans_x,selection,String),
    \+string_to_num(String,_).
invalid_point(non_number) :- object(@trans_y),get(@trans_y,selection,String),
    \+string_to_num(String,_).
invalid_point(too_deep) :- object(@new_depth), max_vehicle_depth(Deepest),
    get(@new_depth,selection,Depth), Depth > Deepest.
invalid_point(bottom_hit) :- object(@new_depth), max_area_depth(Bottom),
    get(@new_depth,selection,Depth), Depth > Bottom.
invalid_point(too_shallow) :- object(@new_depth),
    get(@new_depth,selection,Depth), Depth < 0.
invalid_point(out_of_area) :- object(@new_x), get(@new_x,selection,X),
    op_area(X1,_,X2,_), (X < X1; X > X2).
invalid_point(out_of_area) :- object(@new_y), get(@new_y,selection,Y),
    op_area(_,Y1,_,Y2), (Y < Y1; Y > Y2).
invalid_point(bottom_hit) :- object(@new_depth),
    get(@new_depth,selection,Depth),
    object(@new_x), get(@new_x,selection,X), object(@new_y),
    get(@new_y,selection,Y),
    area_depth(X1,Y1,X2,Y2,Area_depth), Y >= Y1, Y <= Y2, X >= X1, X <= X2,
    Area_depth < Depth.
invalid_point(out_of_area) :- object(@trans_x), get(@trans_x,selection,X),
    op_area(X1,_,X2,_), (X < X1; X > X2).
invalid_point(out_of_area) :- object(@trans_y), get(@trans_y,selection,Y),
    op_area(_,Y1,_,Y2), (Y < Y1; Y > Y2).
invalid_point(bottom_hit) :- object(@new_depth),
    get(@new_depth,selection,Depth),
    object(@trans_x), get(@trans_x,selection,X), object(@trans_y),
    get(@trans_y,selection,Y),
    area_depth(X1,Y1,X2,Y2,Area_depth), Y >= Y1, Y <= Y2, X >= X1, X <= X2,
    Area_depth < Depth.

/* Error Reporting Routines for Phase Errors and Mission Errors */

/* Modify or Delete phase selected when no phases have been specivied */
invalid_option_report(_) :- destroy_error_objects, fail.
invalid_option_report(Error) :- error_code(Error,Message),
    new(@error_window1,dialog('Invalid Option')),
    new(OK,button('OK',message(@prolog,destroy_error_objects))),
    new(Label,label), send(Label,selection,Message),
    send(@error_window1,append,Label),

```

```

send(OK,below,Label), send(@error_window1,open).

/* There is an error in the phase specification */
invalid_phase_report(_) :- destroy_error_objects, fail.
invalid_phase_report(Code) :- error_code(Code,Message),
    new(@error_window2,dialog('Invalid Phase')),
    new(OK,button('OK',message(@prolog,destroy_error_objects))),
    new(Label1,label),
    send(Label1,selection,'PHASE ERROR: THE SPECIFIED PHASE IS INVALID'),
    new(Label2,label), send(Label2,selection,Message),
    send(@error_window2,append,Label1), send(Label2,below,Label1),
    send(OK,below,Label2), send(@error_window2,open).

/* There is a loop in the mission specification, no further parsing can occur
/* until the loop is eliminated */
invalid_mission_report(_,_) :- destroy_error_objects, fail.
invalid_mission_report(Phase,Code) :- error_code(Code,Message),
    new(@error_window3,dialog('Mission Error')),
    new(OK,button('OK',message(@prolog,destroy_error_objects))),
    new(Label1,label),
    send(Label1,selection,'MISSION ERROR IN PHASE: '),
    new(Label2,label), send(Label2,selection,Phase),
    new(Label3,label), send(Label3,selection,Message),
    send(@error_window3,append,Label1), send(Label2,right,Label1),
    send(Label3,below,Label1), send(OK,below,Label3),
    send(@error_window3,open).

/* There is at least one error associated with a phase of the mission
/* All errors associated with a phase are displayed at once */
phase_error_report(_,[LastError]) :- object(LastError),
    new(OK,button('OK',message(@prolog,destroy_error_objects))),
    send(OK,below,LastError),
    send(@error_window4,open).
phase_error_report(Phase,Errors) :- \+object(@error_window4),
    new(@error_window4,dialog('Mission Errors')),
    new(Label1,label), send(Label1,selection,'MISSION ERRORS IN PHASE: '),
    new(Label2,label), send(Label2,selection,Phase),
    send(@error_window4,append,Label1),
    send(Label2,right,Label1),
    phase_error_report(Phase,[Label1 | Errors]).
phase_error_report(Phase,[Previous_error, Current_error | Rest]) :-
    error_code(Current_error,Message),
    new(Label,label), send(Label,selection,Message),
    send(Label,below,Previous_error),
    phase_error_report(Phase,[Label | Rest]).

/* Error message when errors are detected in points specified */
/* in the means end mission generation facility */
invalid_point_report(_) :- destroy_error_objects, fail.
invalid_point_report(Code) :- error_code(Code,Message),
    new(@error_window2,dialog('Invalid Phase')),
    new(OK,button('OK',message(@prolog,destroy_error_objects))),
    new(Label1,label),
    send(Label1,selection,
        'POINT ERROR: ONE OF THE SPECIFIED POINTS IS INVALID'),
    new(Label2,label), send(Label2,selection,Message),
    send(@error_window2,append,Label1), send(Label2,below,Label1),
    send(OK,below,Label2), send(@error_window2,open).

```

```

/* Codes to match error names to an output message */
error_code(no_phases, 'There are no specified phases in memory').
error_code(no_me_modify,
    'You cannot modify a mission generated by the Means Ends Help Facility').
error_code(no_name, 'You did not specify a name for the phase').
error_code(duplicate_phase, 'A phase by that name has already been specified').
error_code(no_successor, 'You did not specify one of the successor phases').
error_code(non_number, 'You have entered a non number in a numerical field').
error_code(bad_heading, 'The heading must be between 0 and 360 degrees').
error_code(too_deep, 'The depth you have entered is too deep for the vehicle').
error_code(bottom_hit, 'The depth you specified is too deep for this area').
error_code(too_shallow, 'Depth must be positive').
error_code(out_of_area, 'The specified point is outside the designated operating area').
error_code(unreachable, 'Phase not reachable from start phase').
error_code(loop, 'Phase reachable from itself').
error_code(dangle, 'Successor Phase Undefined').
error_code(short_timer, 'Specified time-out inadequate for phase completion').
error_code(no_complete, 'No mission completion criteria specified').
error_code(no_start, 'No mission starting phase specified').
error_code(no_file, 'You did not specify an output file').
error_code(hover_for_recovery,
    'Tube recovery must be preceded by a hover phase').
error_code(long_recovery, 'Hover preceeding tube recovery too distant').
error_code(short_recovery,
    'Hover preceeding tube recovery too close for safety').
error_code(not_mission_complete,
    'Successful tube recovery must be last phase').
error_code(displaced_search,
    'Search phase not preceded by hover at the same location').

```

```

% File   : resets.pl
%
% Author : Duane Davis, Brad Leonhardt
%
% Project: Pheonix AUV Strategic Level Mission Generation Expert System
%
% Purpose: This file contains routines for destroying objects and retracting
%          facts as required. Explicitly named Prowindows objects must
%          be destroyed before a new object with the same name can be
%          created.
%
% System : Quintus Prolog 3.2 with pwxpce 3.0 (Prowindows)
%
% Use    : ai4/usr/work3/auv/strategic> newprowin
%          ?- [mission_expert].
%          ?- go.
%
% Date   : 29 April 96

```

```

/* Destroy all Explicitly Named Objects in the Initial Window */
destroy_start_window_objects :- free(@start_dialog),
    free(@start_dialog),
    free(@file_name),
    free(@path_x), free(@path_y),
    free(@path_x1), free(@path_y1),
    free(@picture), free(@bmp).

/* Destroy Menu Containing Phase Graph Summary */
destroy_phase_graph_window :- free(@phase_dialog),
    free(@phases),
    free(@c_succ), free(@a_succ).

```

```

destroy_phase_summary_menu :- free(@phase_dialog).

/* Destroy Menu For Entering Type of Phase */
destroy_phase_type_entry_objects :- free(@type_dialog).

/* Destroy Menus For Entering Individual Phases */
destroy_phase_entry_objects :- free(@parameter_dialog),
    free(@phase_name),
    free(@new_depth),
    free(@time_out),
    free(@heading),
    free(@done),
    free(@reset),
    free(@new_x),
    free(@new_y),
    free(@pointdialog),
    free(@medialog),
    free(@delete_dialog),
    free(@modify_dialog),
    free(@me_route),
    abolish(current_phase/1),
    abolish(abort_successor/1),
    abolish(complete_successor/1),
    object(@mouse_click_path), send(@mouse_click_path,clear),
    fail.
destroy_phase_entry_objects.

/* Destroy Unspecified Phase Entry Dialog Box Objects */
destroy_unspecified_phase_entry_objects :-
    free(@ok1),
    free(@name_entry),
    free(@name).

/* Destroy Code Selection Objects Destroys the Window used to */
/* Specify Desired Output Language of Strategic Level Code */
destroy_code_selection_objects :- free(@code_dialog).

/* Destroy Initialization Entry Objects Destroys Objects in */
/* The Initialization Script Generation Window */
destroy_initialization_entry_objects :-
    free(@initialize_dialog),
    free(@posture_x), free(@posture_y), free(@posture_z),
    free(@posture_phi), free(@posture_theta), free(@posture_psi),
    free(@dt1x), free(@dt1y), free(@dt1z),
    free(@dt2x), free(@dt2y), free(@dt2z),
    free(@gyro_error).

/* Destroy ME Objects destroys all objects associated with */
/* The Means Ends Analysis help facility */
destroy_me_objects :-
    destroy_search_point_and_tube_objects,
    destroy_me_soln_objects,
    destroy_main_me_menu_objects.

/* Destroy Main ME Menu Objects destroys all objects */
/* associated with the top level means ends help menu */
destroy_main_me_menu_objects :-
    free(@medialog),
    free(@startx), free(@starty), free(@startz),
    free(@endx), free(@endy), free(@endz).

/* Destroy Search Point and Tube Objects destroys all objects */
/* associated with the window in which search points are entered */
/* or the window in which recovery tube data is entered */
destroy_search_point_and_tube_objects :-

```

```

    free(@pointdialog), free(@tubedialog),
    free(@trans_x), free(@trans_y),
    free(@new_x), free(@new_y), free(@new_depth), free(@heading).

/* Destroy ME Soln Objects destroys all objects associated with */
/* window in which the means ends solution is displayed */
destroy_me_soln_objects :-
    free(@solutiondialog),
    free(@solutionlabel).

clear_search_points :- abolish(searchpoint/1).

clear_tube_data :- abolish(tube_data/4).

/* quit destroys all objects that are currently in existence */
quit :- destroy_start_window_objects,
        destroy_phase_graph_window,
        reset_to_main_menu.

/* Reset to Main Menu Destroys Menus All Menus Except the Initial One */
reset_to_main_menu :- abolish(current_phase/1),
                      abolish(abort_successor/1),
                      abolish(complete_successor/1),
                      destroy_phase_entry_objects,
                      destroy_phase_type_entry_objects,
                      destroy_unspecified_phase_entry_objects,
                      destroy_error_objects,
                      destroy_code_selection_objects,
                      destroy_initialization_entry_objects,
                      destroy_me_objects.

reset_to_main_menu.

/* Destroy error dialog box */
destroy_error_objects :- free(@error_window1),
                        free(@error_window2),
                        free(@error_window3),
                        free(@error_window4),
                        free(@ok).

/* Reset For ME Solution destroys the means ends solution */
/* menu and all of the objects within it, and abolishes */
/* all facts associated with a means ends solution so that */
/* the next means ends solution can be computed and displayed */
reset_for_me_solution :- destroy_me_soln_objects, erase_paths,
                        abolish(phase/5), abolish(start_phase/1), asserta(start_phase(1)),
                        abolish(next_phase/1), asserta(next_phase(1)).

/* Erase Paths clears the means end solution path and the */
/* mouse click path from the map picture */
erase_paths :- \+object(@mouse_click_path),
              new(@mouse_click_path,path), send(@mouse_click_path,pen,2),
              send(@picture,display,@mouse_click_path), fail.
erase_paths :-
    free(@me_route),
    send(@mouse_click_path,clear).

```

```

% File    : utilities.pl
%
% Author  : Duane Davis, Brad Leonhardt
%
% Project: Pheonix AUV Strategic Level Mission Generation Expert System
%
% Purpose: This file contains routines for miscellaneous operations required
%          by various parts of the Mission Generation Expert System.
%          Operations include list processing, mathematical operations,
%          and menu construction.
%
% System  : Quintus Prolog 3.2 with pwxpce 3.0 (Prowindows)
%
% Use     : ai4/usr/work3/auv/strategic> newprowin
%          ?- [mission_expert].
%          ?- go.
%
% Date    : 29 April 96

```

```

successor(Phase1,Phase2) :- phase(Phase1,_,_,Phase2,_).
successor(Phase1,Phase2) :- phase(Phase1,_,_,_,Phase2).

reachable(Phase1,Phase2) :- successor(Phase1,Phase2).
reachable(Phase1,Phase2) :- successor(Phase1,Phase3), reach-
able(Phase3,Phase2).

transit_distance(Phase,Distance) :- phase(Phase,Type1,[X1, Y1 | L1],_,_),
\+same(Type1,'GPS Fix'), \+same(Type1,'Depth Change'),
position_predecessor(Phase,Predecessor),
phase(Predecessor,_,[X2, Y2 | L2],_,_),
distance(X1,Y1,X2,Y2,Distance).
transit_distance(Phase,Distance) :- phase(Phase,Type1,[X1, Y1 | L],_,_),
\+same(Type1,'GPS Fix'), \+same(Type1,'Depth Change'),
start_phase(Start), start_position(X2,Y2,D), same(Start,Phase),
distance(X1,Y1,X2,Y2,Distance).

max_transit_distance(Phase,Distance) :-
bagof(Dist,transit_distance(Phase,Dist),DList),
sort(DList,SortedList),last(SortedList,Distance).

position_predecessor(Phase,Predecessor) :-
(phase(Predecessor,Type,_,Phase,_); phase(Predecessor,Type,_,_,Phase)),
\+same(Type,'GPS Fix'), \+same(Type,'Depth Change').
position_predecessor(Phase,Predecessor) :-
(phase(Phase2,Type,_,Phase,_); phase(Phase2,Type,_,_,Phase)),
(same(Type,'GPS Fix'); same(Type,'Depth Change')),
position_predecessor(Phase2,Predecessor).

distance(X1,Y1,X2,Y2,Distance) :-
X_plus_Y_Squared is (X2-X1)*(X2-X1) + (Y2-Y1)*(Y2-Y1),
sqrt(X_plus_Y_Squared,Distance).

approach_point(Xtube,Ytube,Theta,Xapproach,Yapproach) :-
deg_to_rad(Theta,Theta_rad),
cos(Theta_rad,Cos_theta_rad), sin(Theta_rad,Sin_theta_rad),
XapproachFloat is Xtube - 15.0 * Cos_theta_rad,
round(XapproachFloat,Xapproach),
YapproachFloat is Ytube - 15.0 * Sin_theta_rad,
round(YapproachFloat,Yapproach).

perpendicular_point(Xtube,Ytube,Theta,Xperp,Yperp) :-
deg_to_rad(Theta,Theta_rad),
cos(Theta_rad,Cos_theta_rad), sin(Theta_rad,Sin_theta_rad),
XperpFloat is Xtube - 15.0 * Sin_theta_rad,
round(XperpFloat,Xperp),

```

```

    YperpFloat is Ytube + 15.0 * Cos_theta_rad,
    round(YperpFloat,Yperp).

deg_to_rad(Deg,Rad) :- Rad is Deg * 3.1415927 / 180.0.

same(X,X).

string_to_num(String,Num) :- name(String,Asc),name(Num,Asc),number(Num).

last([X],X) :- !.
last([X | L],Y) :- last(L,Y).

concatenate(S1,S2,S) :- name(S1,AS1), name(S2,AS2),
    append(AS1,AS2,AS),name(S,AS).

list_concat([], '').
list_concat([X|L],S) :- list_concat(L,Part), concatenate(X,Part,S).

pad_to_30(String,String) :- name(String,List), length(List,Length),
    Length >= 30, !.
pad_to_30(String,New_string) :- concatenate(String,' ',Sub_string),
    pad_to_30(Sub_string,New_string).

member(X,[X|L]).
member(X,[Y|L]) :- member(X,L).

same_members([],[]).
same_members([X1 | L1],L2) :- member(X1,L2), delete(X1,L2,NewL2),
    same_members(L1,NewL2).

subset([],L).
subset([X|L],L2) :- singlemember(X,L2), subset(L,L2).

singlemember(X,[X|L]) :- !.
singlemember(X,[Y|L]) :- singlemember(X,L).

union([],L,L).
union([X|L1],L2,L3) :- singlemember(X,L2), !, union(L1,L2,L3).
union([X|L1],L2,[X|L3]) :- union(L1,L2,L3).

deleteitems([],L,L).
deleteitems([X|L],L2,L3) :- delete(X,L2,L4), deleteitems(L,L4,L3).

delete(X,[],[]).
delete(X,[X|L],M) :- !, delete(X,L,M).
delete(X,[Y|L],[Y|M]) :- delete(X,L,M).

construct_menu(Menu,Items,Callback) :- member(MenuItem,Items),
    send(Menu,append,menu_item(MenuItem,message(@prolog,Callback,MenuItem))),
    fail.
construct_menu(_,_,_).

/* Automatic Start of Program Upon File Load */

:- go.

/* End of File mission_expert.pl */

```

```

/*****
/*
Program:          AUV strategic level program

Authors:         Brad Leonhardt Duane Davis

Revised:         1 August 96

System:          SUN Voyager Solaris 2.4 OS; SGI Irix 5.3
Compiler:        Sun C; IRIX cc

Compilation:     cc mission_pl.c -o mission_pl

Invocation:      mission_pl [normally invoked automatically by expert system]

[ input file:    command_strings]
[output file:    name is specified by user and contained in command_strings]

This code is used to create Prolog code to run the Phoenix Autonomous Vehicle
It can be run with the prolog mission_expert system which creates a data
file for use in the mission generator.

*****/

#include <string.h>
#include <stdio.h>
#include <ctype.h>

#define DATA_FILE "command_strings" /* Expert System Output */

/*****
/*          function prototypes          */
*****/

int      main                                ();
void     depth_change                       ();
void     hoverpoint                        ();
void     waypoint                          ();
void     sonar_search                      ();
void     get_gps_fix                       ();
void     rotate_sonar_search               ();
void     course                           ();
void     wait                             ();
void     tube_recovery                     ();
void     parse_command                    ();
void     time_out                         ();
void     next_phase                       ();
void     fail_phase                       ();

FILE * file_ptr;
FILE * out_file_ptr;
FILE * standalone_out_file_ptr;

char command           [200];
char out_file_name     [200];
char standalone_out_file_name [200];
char shell_cmd         [200];
char phase_name        [200];
char nextphase         [200];
char failphase         [200];
char cmd               [200];

int variable1,variable2,variable3,variable4,variable5;
int i;

```



```

/*****
/* Parse command breaks input string into variables to be used in program
*/
*****/

void parse_command (cmd)
char * cmd;
{
    sscanf(cmd, "%s %s %s %s %d %d %d %d %d",
           command, phase_name, nextphase, failphase,
           &variable1, &variable2, &variable3, &variable4, &variable5);

/* printf ("COMMAND %s PHASE NAME %s VAR1 %d VAR2 %d VAR3 %d VAR4 %d VAR5 %d",
   command, phase_name, nextphase, failphase,
   variable1, variable2, variable3, variable4, variable5);
*/
}

/*****/

int main ()
{
    char cmd [200]; /* input line string of characters */

/* Open file generated from expert system */

    if ((file_ptr = fopen (DATA_FILE, "r")) == ((FILE *) 0))
    {
        printf ("input_read_path: file open failure!\n");
        return;
    }

    fgets (cmd, 200, file_ptr); /* Read a line of data */

    parse_command (cmd);

/* Create an output file for mission and copy mission controller data */

    if (strcmp (command, "file_name") == 0)
    {
        strcpy(out_file_name, phase_name);
        strcpy(standalone_out_file_name, "standalone_");
        strcat(standalone_out_file_name, phase_name);
        strcat(shell_cmd, "cp controller_pl.script ");
        strcat(shell_cmd, out_file_name);
        printf("Shell command :%s\n", shell_cmd);
        system(shell_cmd);
    }

/* Create output file for standalone strategic level code */
    strcpy(shell_cmd, "cp controller_standalone_pl.script ");
    strcat(shell_cmd, "standalone_");
    strcat(shell_cmd, out_file_name);
    printf("Shell command :%s\n", shell_cmd);
    system(shell_cmd);

    printf("Output files created\n");

    if (((out_file_ptr = fopen (out_file_name, "a")) == ((FILE *) 0)) ||
        ((standalone_out_file_ptr = fopen (standalone_out_file_name, "a")) ==
         ((FILE *) 0)))
    {

```

```

    printf("INVALID INPUT FILE NAME %s\n",out_file_name);
    fclose (file_ptr);
    return(0);
}

fprintf (out_file_ptr,
        "\t\t\t asserta(current_phase(%s)),\n",nextphase);
fprintf (out_file_ptr, "\t\t\t asserta(succeed(0)),\n");
fprintf (out_file_ptr, "\t\t\t asserta(abort(0)).\n\n");

fprintf (standalone_out_file_ptr,
        "\t\t\t asserta(current_phase(%s)),\n",nextphase);
fprintf (standalone_out_file_ptr, "\t\t\t asserta(succeed(0)),\n");
fprintf (standalone_out_file_ptr, "\t\t\t asserta(abort(0)).\n\n");

while ((fgets (cmd, 200, file_ptr) != NULL)) /* read next line */
{
    printf("Input file string %s\n",cmd);
    parse_command(cmd);

    fprintf (out_file_ptr, "%s\n\n\n",phase_name);
    fprintf (out_file_ptr, "execute_phase(%s)\n",phase_name);
    fprintf (out_file_ptr, "\t\t\t :- nl,printsc('PHASE");
    fprintf (out_file_ptr, " %s STARTED.').\n",phase_name);

    fprintf (standalone_out_file_ptr, "%s\n\n\n",phase_name);
    fprintf (standalone_out_file_ptr, "execute_phase(%s)\n",phase_name);
    fprintf (standalone_out_file_ptr, "\t\t\t :- nl,printsc('PHASE");
    fprintf (standalone_out_file_ptr, " %s STARTED.').\n",phase_name);

    /* Determine which template to use */

    if (strcmp (command,"depth_change") == 0)
        depth_change ();

    else if (strcmp (command,"hoverpoint") == 0)
        hoverpoint ();

    else if (strcmp (command,"waypoint") == 0)
        waypoint ();

    else if (strcmp (command,"sonar_search") == 0)
        sonar_search ();

    else if (strcmp (command,"get_gps_fix") == 0)
        get_gps_fix ();

    else if (strcmp (command,"rotate_sonar_search") == 0)
        rotate_sonar_search ();

    else if (strcmp (command,"course") == 0)
        course ();

    else if (strcmp (command,"wait") == 0)
        wait ();

    else if (strcmp (command,"tube_recovery") == 0)
        tube_recovery ();

    else
    {
        printf("INVALID PHASE INPUT %s\n",command);
        fprintf (file_ptr, "\n");
        fclose (file_ptr);
        fprintf (out_file_ptr, "\n");
        fclose (out_file_ptr);
        fprintf (standalone_out_file_ptr, "\n");
        fclose (standalone_out_file_ptr);
        return(0);
    }
}

```



```

/*
Program:          AUV strategic level program

Authors:         Brad Leonhardt Duane Davis

Revised:         14 May 96

System:          SUN Voyager Solaris 2.4 OS; SGI Irix 5.3
Compiler:        Sun C; IRIX cc

Compilation:     cc mission_cpp.c -o mission_cpp

Invocation:      mission_cpp [normally invoked automatically by expert system]

[ input file:    command_strings]
[output file:    mission_graph.C]

```

This code is used to create Prolog code to run the Phoenix Autonomous Vehicle
It can be run with the mission_expert prolog expert system which creates a data
file for use in the mission generator.

```

/*****

#include <string.h>
#include <stdio.h>
#include <ctype.h>

#define DATA_FILE "command_strings"      /* Expert System Output */

/*****
/*          function prototypes
/*
/*****

int      main
void      depth_change          ();
void      hoverpoint           ();
void      waypoint              ();
void      sonar_search          ();
void      get_gps_fix           ();
void      rotate_sonar_search   ();
void      course                ();
void      wait                  ();
void      tube_recovery         ();
void      parse_command         ();
void      successors            ();

FILE * file_ptr;
FILE * out_file_ptr;

char command      [200];
char out_file_name [200];
char shell_cmd    [200];
char phase_name   [200];
char nextphase    [200];
char failphase    [200];
char cmd          [200];

int variable1,variable2,variable3,variable4,variable5;
int i;

```

```

/*****
/* Parse command breaks input string into variables to be used in program */
*****/

void parse_command (cmd)
char * cmd;
{
    sscanf(cmd, "%s %s %s %s %d %d %d %d %d",
           command,phase_name,nextphase,failphase,
           &variable1,&variable2,&variable3,&variable4,&variable5);

/* printf ("COMMAND %s PHASE NAME %s VAR1 %d VAR2 %d VAR3 %d VAR4 %d VAR5 %d",
   command,phase_name,nextphase,failphase,
   variable1,variable2,variable3,variable4,variable5);
*/
}

/*****/

int main ()
{
    char cmd [200];          /*      input line string of characters      */

/*      Open file generated from expert system      */

    if ((file_ptr = fopen (DATA_FILE, "r")) == ((FILE *) 0))
    {
        printf ("input_read_path:  file open failure!\n");
        return;
    }

    fgets (cmd,200,file_ptr); /*      Read a line of data      */

    parse_command (cmd);

/*Create an output file for mission and copy mission controller data */
    if (strcmp (command,"file_name") == 0)
    {
        strcpy(out_file_name,phase_name);
        strcat(shell_cmd,"cp controller_cpp.script ");
        strcat(shell_cmd,out_file_name);
        printf("Shell command :%s\n",shell_cmd);
        system(shell_cmd);
    }

    printf("Output file created\n");

    if ((out_file_ptr = fopen (out_file_name, "a")) == ((FILE *) 0))
    {
        printf("INVALID OUTPUT FILE NAME  %s\n",out_file_name);
        fclose (file_ptr);
        return(0);
    }

    while ((fgets (cmd,200, file_ptr) != NULL)) /*      read next line      */
    {
        printf("Input file string %s\n",cmd);
        parse_command(cmd);

/*      Determine which template to use      */

        if (strcmp (command,"depth_change") == 0)
            depth_change ();

        else if (strcmp (command,"hoverpoint") == 0)

```

```

        hoverpoint ();

    else if (strcmp (command,"waypoint") == 0)
        waypoint ();

    else if (strcmp (command,"sonar_search") == 0)
        sonar_search ();

    else if (strcmp (command,"get_gps_fix") == 0)
        get_gps_fix ();

    else if (strcmp (command,"rotate_sonar_search") == 0)
        rotate_sonar_search ();

    else if (strcmp (command,"course") == 0)
        course ();

    else if (strcmp (command,"wait") == 0)
        wait ();

    else if (strcmp (command,"tube_recovery") == 0)
        tube_recovery ();

    else
    {
        printf("INVALID PHASE INPUT %s\n",command);
        fprintf (file_ptr, "\n");
        fclose (file_ptr);
        fprintf (out_file_ptr, "\n");
        fclose (out_file_ptr);
        return(0);
    }
}

fprintf (file_ptr, "\n");
fclose (file_ptr);

/* Pass 2 through data file generated by expert system */
fprintf (out_file_ptr, "\n\n\t/* Link Phases Into DFA Graph */\n");
if ((file_ptr = fopen (DATA_FILE, "r")) == ((FILE *) 0))
{
    printf ("input_read_path: file open failure!\n");
    return;
}

fgets (cmd,200,file_ptr); /*          Read a line of data          */
parse_command (cmd);

if (strcmp (command,"file_name") == 0)
{
    fprintf (out_file_ptr,
        "\tphinitialize->specifySuccessors(ph%s,phmission_abort);\n",
        nextphase);
}

while ((fgets (cmd,200, file_ptr) != NULL)) /*      read next line      */
{
    parse_command (cmd);
    successors ();
}

fprintf (out_file_ptr, "\n\treturn (phinitialize);\n\n");

```

```

        fprintf (out_file_ptr, "} // buildMissionGraph\n");

        fprintf (file_ptr, "\n");
        fclose (file_ptr);
        fprintf (out_file_ptr, "\n");
        fclose (out_file_ptr);
    }

    /*****
    /*  Functions that produce the C++ code for the various types of phases  */
    *****/

    void depth_change ()
    {
        fprintf (out_file_ptr, "\tDepthChange\t*ph%s = ", phase_name);
        fprintf (out_file_ptr, "new DepthChange (%d,%d);\n", variable2, variable1);
    }

    void hoverpoint ()
    {
        fprintf (out_file_ptr, "\tHover\t\t*ph%s = ", phase_name);
        fprintf (out_file_ptr, "new Hover (%d,%d,%d,%d);\n",
            variable2, variable3, variable4, variable5, variable1);
    }

    void waypoint ()
    {
        fprintf (out_file_ptr, "\tTransit\t\t*ph%s = ", phase_name);
        fprintf (out_file_ptr, "new Transit (%d,%d,%d,%d);\n",
            variable2, variable3, variable4, variable1);
    }

    void sonar_search ()
    {
        fprintf (out_file_ptr, "\tRotateSearch\t*ph%s = ", phase_name);
        fprintf (out_file_ptr, "new RotateSearch (%d,%d,%d,%d);\n",
            variable2, variable3, variable4, variable5, variable1);
    }

    void get_gps_fix ()
    {
        fprintf (out_file_ptr, "\tGPSFix\t\t*ph%s = ", phase_name);
        fprintf (out_file_ptr, "new GPSFix (%d);\n", variable1);
    }

    void rotate_sonar_search ()

```

```

{
    fprintf (out_file_ptr, "\tSonarSearch\t*ph%s = ", phase_name);
    fprintf (out_file_ptr, "new SonarSearch (%d,%d,%d,%d,%d);\n",
        variable2, variable3, variable4, variable5, variable1);
}

void course ()
{
    fprintf (out_file_ptr, "\tCourseChange\t*ph%s = ", phase_name);
    fprintf (out_file_ptr,
        "new CourseChange (%d,%d);\n", variable2, variable1);
}

void wait ()
{
    fprintf (out_file_ptr, "\tWait\t\t*ph%s = ", phase_name);
    fprintf (out_file_ptr, "new Wait (%d);\n", variable1);
}

void tube_recovery ()
{
    fprintf (out_file_ptr, "\tTubeRecovery\t*ph%s = ", phase_name);
    fprintf (out_file_ptr, "new TubeRecovery (%d,%d,%d,%d,%d);\n",
        variable2, variable3, variable4, variable5, variable1+250);
}

void successors ()
{
    fprintf (out_file_ptr, "\tph%s->specifySuccessors(ph%s,ph%s);\n",
        phase_name, nextphase, failphase);
}

```


LIST OF REFERENCES

Brutzman, D. P., *A Virtual World for an Autonomous Undersea Vehicle*, Ph.D. Dissertation, Naval Postgraduate School, Monterey California, December 1994. Available at <http://www.stl.nps.navy.mil/~brutzman/dissertation>

Byrnes, R., *The Rational Behavior Model: A Multi-Paradigm, Tri-Level Software Architecture for the Control of Autonomous Vehicles*, Ph.D. Dissertation, Naval Postgraduate School, Monterey California, March 1993.

Davis, D. T., *Precision Maneuvering and Control of the Phoenix Autonomous Underwater Vehicle for Entering a Recovery Tube*, Master's Thesis, Naval Postgraduate School, Monterey California, September 1996. Available at http://www.cs.nps.navy.mil/research/auv/auv_thesisarchive.html

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center.....2
8725 John J. Kingman Rd., STE 0944
Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library.....2
Naval Postgraduate School
411 Dyer Road
Monterey, CA 93943-5101

3. Dr. Robert B. McGhee, Professor.....1
Computer Science Department, Code CS/Mz
Naval Postgraduate School
Monterey, CA 93943-5100

4. Dr. Don Brutzman, Associate Professor.....1
Undersea Warfare, Code UW/Br
Naval Postgraduate School
Monterey, CA 93943-5100

5. Dr. Anthony Healey, Professor.....1
Mechanical Engineering Department, Code ME/Hy
Naval Postgraduate School
Monterey, CA 93943-5000

6. CDR Michael Holden, USN1
Computer Science Department, Code CS/Hm
Naval Postgraduate School
Monterey, CA 93943-5100

7. Caroline Deltheil.....1
Centre Technique des Systemes Navals (CTSN)
Dissuasion Lutte Sous-Marine (DLSN)
DGA/DCN
BP-28-83800
Toulon Naval, France

8. Dr. John Leonard, Research Engineer1
Underwater Vehicles Laboratory,
Massachusetts Institute of Technology
Sea Grant College Program
292 Main Street

Cambridge, MA 02139

9. LT Duane T. Davis1
1008 Greenland Circle
South Charleston, WV 25309